

University of Oxford

College of Computer Science

Elzein Salah Elzein

22/12/2021

PHD Report Draft

Written By Elzein Salah Elzein

Natural Language Generation for Social Robotics

Contents (Section)	Page
1. Introduction	2
2. Literature Review	57
3. Social Robots	276
4. Research Overview	298
5. Methodology	405
6. Related Work and Discussion	577
7. Conclusion and Future Work	996
8. References	1087
9. Appendices	1110

1. Introduction



Classical Approaches

1	Classical Approaches to Natural Language Processing <i>Robert Dale</i>	3
	Context · The Classical Toolkit · Conclusions · Reference	
2	Text Preprocessing <i>David D. Palmer</i>	9
	Introduction · Challenges of Text Preprocessing · Tokenization · Sentence Segmentation ·	
	Conclusion · References	
3	Lexical Analysis <i>Andrew Hippiusley</i>	31
	Introduction · Finite State Morphology · Finite State Morphology · “Difficult”	
	Morphology and Lexical Analysis · Paradigm-Based Lexical Analysis · Concluding Remarks ·	
	Acknowledgments · References	
4	Syntactic Parsing <i>Peter Ljunglöf and Mats Wirén</i>	59
	Introduction · Background · The Cocke–Kasami–Younger Algorithm · Parsing as	
	Deduction · Implementing Deductive Parsing · LR Parsing · Constraint-based	
	Grammars · Issues in Parsing · Historical Notes and Outlook · Acknowledgments ·	
	References	
5	Semantic Analysis <i>Cliff Goddard and Andrea C. Schalley</i>	93
	Basic Concepts and Issues in Natural Language Semantics · Theories and Approaches to	
	Semantic Representation · Relational Issues in Lexical Semantics · Fine-Grained	
	Lexical-Semantic Analysis: Three Case Studies · Prospectus and “Hard Problems” ·	
	Acknowledgments · References	
6	Natural Language Generation <i>David D. McDonald</i>	121
	Introduction · Examples of Generated Texts: From Complex to Simple and Back Again	
	· The Components of a Generator · Approaches to Text Planning · The Linguistic	

1

Classical Approaches to Natural Language Processing

1.1	Context	3
1.2	The Classical Toolkit	4
	TextPreprocessing · LexicalAnalysis · SyntacticParsing · Semantic Analysis · Natural Language Generation	
1.3	Conclusions.....	6
	Reference	7

Robert Dale
Macquarie University

1.1 Context

The first edition of this handbook appeared in 2000, but the project that resulted in that volume in fact began 4 years earlier, in mid-1996. When Hermann Moisl, Harold Somers, and I started planning the content of the book, the field of natural language processing was less than 10 years into what some might call its “statistical revolution.” It was still early enough that there were occasional signs of friction between some of the “old guard,” who hung on to the symbolic approaches to natural language processing that they had grown up with, and the “young turks,” with their new-fangled statistical processing techniques, which just kept gaining ground. Some in the old guard would give talks pointing out that there were problems in natural language processing that were beyond the reach of statistical or corpus-based methods; meanwhile, the occasional young Turk could be heard muttering a variation on Fred Jelinek’s 1988 statement that “whenever I fire a linguist our system performance improves.” Then there were those with an eye to a future peaceful coexistence that promised jobs for all, arguing that we needed to develop hybrid techniques and applications that integrated the best properties of both the symbolic approaches and the statistical approaches.

At the time, we saw the handbook as being one way of helping to catalog the constituent tools and techniques that might play a role in such hybrid enterprises. So, in the first edition of the handbook, we adopted a tripartite structure, with the 38 book chapters being fairly evenly segregated into Symbolic Approaches to NLP, Empirical Approaches to NLP, and NLP based on Artificial Neural Networks. The editors of the present edition have renamed the Symbolic Approaches to NLP part as Classical Approaches: that name change surely says something about the way in which things have developed over the last 10 years or so. In the various conferences and journals in our field, papers that make use of statistical techniques now very significantly outnumber those that do not. The number of chapters in the present edition of the handbook that focus on these “classical” approaches is half the number that focus on the

empirical and statistical approaches. But these changes should not be taken as an indication that the earlier-established approaches are somehow less relevant; in fact, the reality is quite the opposite, as

the incorporation of linguistic knowledge into statistical processing becomes more and more common. Those who argue for the study of the classics in the more traditional sense of that word make great claims for the value of such study: it encourages the questioning of cultural assumptions, allows one to appreciate different cultures and value systems, promotes creative thinking, and helps one to understand the development of ideas. That is just as true in natural language processing as it is in the study of Greek literature.

So, in the spirit of all those virtues that surely no one would question, this part of the handbook provides thoughtful and reflective overviews of a number of areas of natural language processing that are in some sense foundational. They represent areas of research and technological development that have been around for some time; long enough to benefit from hindsight and a measured and more objective assessment than is possible for work that is more recent. This introduction comments briefly on each of these chapters as a way of setting the scene for this part of the handbook as a whole.

1.2 The Classical Toolkit

Traditionally, work in natural language processing has tended to view the process of language analysis as being decomposable into a number of stages, mirroring the theoretical linguistic distinctions drawn between SYNTAX, SEMANTICS, and PRAGMATICS. The simple view is that the sentences of a text are first analyzed in terms of their syntax; this provides an order and structure that is more amenable to an analysis in terms of semantics, or literal meaning; and this is followed by a stage of pragmatic analysis whereby the meaning of the utterance or text in context is determined. This last stage is often seen as being concerned with DISCOURSE, whereas the previous two are generally concerned with sentential matters. This attempt at a correlation between a stratificational distinction (syntax, semantics, and pragmatics) and a distinction in terms of granularity (sentence versus discourse) sometimes causes some confusion in thinking about the issues involved in natural language processing; and it is widely recognized that in real terms it is not so easy to separate the processing of language neatly into boxes corresponding to each of the strata. However, such a separation serves as a useful pedagogic aid, and also constitutes the basis for architectural models that make the task of natural language analysis more manageable from a software engineering point of view.

about semantics and discourse-level processing. But it also reflects the

Nonetheless, the tripartite distinction into syntax, semantics, and pragmatics only serves at best as a starting point when we consider the processing of real natural language texts. A finer-grained decomposition of the process is useful when we take into account the current state of the art in combination with the need to deal with real language data; this is reflected in Figure 1.1.

We identify here the stage of tokenization and sentence segmentation as a crucial first step. Natural language text is generally not made up of the short, neat, well-formed, and well-delimited sentences we find in textbooks; and for languages such as Chinese, Japanese, or Thai, which do not share the apparently easy space-delimited tokenization we might believe to be a property of languages like English, the ability to address issues of tokenization is essential to getting off the ground at all. We also treat lexical analysis as a separate step in the process. To some degree this finer-grained decomposition reflects our current state of knowledge about language processing: we know quite a lot about general techniques for tokenization, lexical analysis, and syntactic analysis, but much less

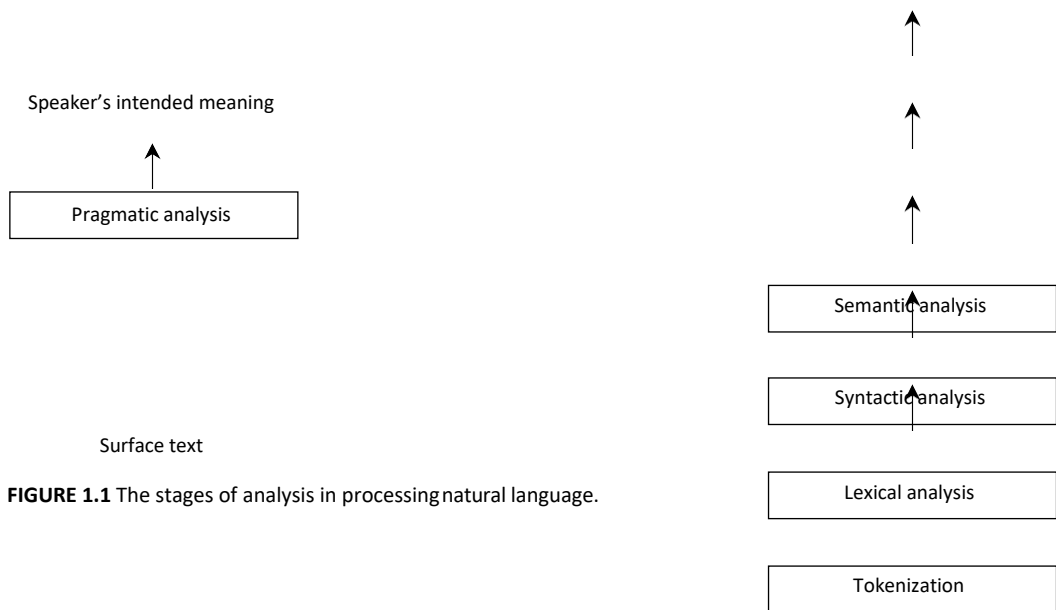


FIGURE 1.1 The stages of analysis in processing natural language.

fact that the known is the surface text, and anything deeper is a representational abstraction that is harder to pin down; so it is not so surprising that we have better-developed techniques at the more concrete end of the processing spectrum.

Of course, natural language analysis is only one-half of the story. We also have to consider natural language generation, where we are concerned with mapping from some (typically nonlinguistic) internal representation to a surface text. In the history of the field so far, there has been much less work on natural language generation than there has been on natural language analysis. One sometimes hears the suggestion that this is because natural language generation is easier, so that there is less to be said. This is far from the truth: there are a great many complexities to be addressed in generating fluent and coherent multi-sentential texts from an underlying source of information. A more likely reason for the relative lack of work in generation is precisely the correlate of the observation made at the end of the previous paragraph: it is relatively straightforward to build theories around the processing of something known (such as a sequence of words), but much harder when the input to the process is more or less left to the imagination. This is the question that causes researchers in natural language generation to wake in the middle of the night in a cold sweat: what does generation start from? Much work in generation is concerned with addressing these questions head-on; work in natural language understanding may eventually see benefit in taking generation's starting point as its end goal.

1.2.1 Text Preprocessing

As we have already noted, not all languages deliver text in the form of words neatly delimited by spaces. Languages such as Chinese, Japanese, and Thai require first that a segmentation process be applied, analogous to the segmentation process that must first be applied to a continuous speech stream in order to identify the words that make up an utterance. As Palmer demonstrates in his chapter, there are significant segmentation and tokenization issues in apparently easier-to-segment languages — such as English — too. Fundamentally, the issue here is that of what constitutes a word; as Palmer shows, there is no easy answer here. This chapter also looks at the problem of sentence segmentation: since so much work in natural language processing views the sentence as the unit of analysis, clearly it is of crucial importance to ensure that, given a text, we can break it into sentence-sized pieces. This turns out not to be so trivial either. Palmer offers a catalog of tips and techniques that will be useful to anyone faced with dealing with real raw text as the input to an analysis process, and provides a healthy reminder that these problems have tended to be idealized away in much earlier, laboratory-based work in natural language processing.

1.2.2 Lexical Analysis

The previous chapter addressed the problem of breaking a stream of input text into the words and sentences that will be subject to subsequent processing. The words, of course, are not atomic, and are themselves open to further analysis. Here we enter the realms of computational morphology, the focus of Andrew Hippius's chapter. By taking words apart, we can uncover information that will be useful at later stages of processing. The combinatorics also mean that decomposing words into their parts, and maintaining rules for how combinations are formed, is much more efficient in terms of storage space than would be the case if we simply listed every word as an atomic element in a huge inventory. And, once more returning to our concern with the handling of real texts, there will always be words missing from any such inventory; morphological processing can go some way toward handling such unrecognized words. Hippius provides a wide-ranging and detailed review of the techniques that can be used to carry out morphological processing, drawing on examples from languages other than English to demonstrate the need for sophisticated processing methods; along the way he provides some background in the relevant

theoretical aspects of phonology and morphology.

1.2.3 Syntactic Parsing

A presupposition in most work in natural language processing is that the basic unit of meaning analysis is the sentence: a sentence expresses a proposition, an idea, or a thought, and says something about some real or imaginary world. Extracting the meaning from a sentence is thus a key issue. Sentences are not, however, just linear sequences of words, and so it is widely recognized that to carry out this task requires an analysis of each sentence, which determines its structure in one way or another. In *NLP* approaches based on generative linguistics, this is generally taken to involve the determining of the syntactic or grammatical structure of each sentence. In their chapter, Ljunglöf and Wirén present a range of techniques that can be used to achieve this end. This area is probably the most well established in the field of *NLP*, enabling the authors here to provide an inventory of basic concepts in parsing, followed by a detailed catalog of parsing techniques that have been explored in the literature.

1.2.4 Semantic Analysis

Identifying the underlying syntactic structure of a sequence of words is only one step in determining the meaning of a sentence; it provides a structured object that is more amenable to further manipulation and subsequent interpretation. It is these subsequent steps that derive a meaning for the sentence in question. Goddard and Schalley's chapter turns to these deeper issues. It is here that we begin to reach the bounds of what has so far been scaled up from theoretical work to practical application. As pointed out earlier in this introduction, the semantics of natural language have been less studied than syntactic issues, and so the techniques described here are not yet developed to the extent that they can easily be applied in a broad-coverage fashion.

After setting the scene by reviewing a range of existing approaches to semantic interpretation, Goddard and Schalley provide a detailed exposition of Natural Semantic Metalanguage, an approach to semantics that is likely to be new to many working in natural language processing. They end by cataloging some of the challenges to be faced if we are to develop truly broad coverage semantic analyses.

1.2.5 Natural Language Generation

At the end of the day, determining the meaning of an utterance is only really one-half of the story of natural language processing: in many situations, a response then needs to be generated, either in natural language alone or in combination with other modalities. For many of today's applications, what is required here is rather trivial and can be handled by means of canned responses; increasingly, however, we are seeing natural language generation techniques applied in the context of more sophisticated back-end systems, where the need to be able to custom-create fluent multi-sentential texts on demand becomes a priority. The generation-oriented chapters in the Applications part bear testimony to the scope here.

In his chapter, David McDonald provides a far-reaching survey of work in the field of natural language generation. McDonald begins by lucidly characterizing the differences between natural language analysis and natural language generation. He goes on to show what can be achieved using natural language generation techniques, drawing examples from systems developed over the last 35 years. The bulk of the chapter is then concerned with laying out a picture of the component processes and representations required in order to generate fluent multi-sentential or multi-paragraph texts, built around the now-standard distinction between text planning and linguistic realization.

1.3 Conclusions

Early research into machine translation was underway in both U.K. and U.S. universities in the mid-1950s,

and the first annual meeting of the Association for Computational Linguistics was in 1963; so, depending

on how you count, the field of natural language processing has either passed or is fast approaching its 50th birthday. A lot has been achieved in this time. This part of the handbook provides a consolidated summary of the outcomes of significant research agendas that have shaped the field and the issues it chooses to address. An awareness and understanding of this work is essential for any modern-day practitioner of natural language processing; as George Santayana put it over a 100 years ago, “Those who cannot remember the past are condemned to repeat it.”

One aspect of computational work not represented here is the body of research that focuses on discourse and pragmatics. As noted earlier, it is in these areas that our understanding is still very much weaker than in areas such as morphology and syntax. It is probably also the case that there is currently less work going on here than there was in the past: there is a sense in which the shift to statistically based work restarted investigations of language processing from the ground up, and current approaches have many intermediate problems to tackle before they reach the concerns that were once the focus of “the discourse community.” There is no doubt that these issues will resurface; but right now, the bulk of attention is focused on dealing with syntax and semantics.* When most problems here have been solved, we can expect to see a renewed interest in discourse-level phenomena and pragmatics, and at that point the time will be ripe for another edition of this handbook that puts classical approaches to discourse back on the table as a source of ideas and inspiration. Meanwhile, a good survey of various approaches can be found in Jurafsky and Martin (2008).

Reference

Jurafsky, D. and Martin, J. H., 2008, *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics*, 2nd edition. Prentice-Hall, Upper Saddle River, NJ.

* A notable exception is the considerable body of work on text summarization that has developed over the last 10 years.

2

Text Preprocessing

2.1	Introduction	9
2.2	Challenges of Text Preprocessing.....	10
	Character-Set Dependence · Language Dependence · Corpus Dependence · Application Dependence	
2.3	Tokenization	15
	Tokenization in Space-Delimited Languages · Tokenization in Unsegmented Languages	
2.4	Sentence Segmentation	22
	Sentence Boundary Punctuation · The Importance of Context · Traditional Rule-Based Approaches · Robustness and Trainability · Trainable Algorithms	
2.5	Conclusion	27
	References	28

David D. Palmer
Autonomy Virage

2.1 Introduction

In the linguistic analysis of a digital natural language text, it is necessary to clearly define the characters, words, and sentences in any document. Defining these units presents different challenges depending on the language being processed and the source of the documents, and the task is not trivial, especially when considering the variety of human languages and writing systems. Natural languages contain inherent ambiguities, and writing systems often amplify ambiguities as well as generate additional ambiguities. Much of the challenge of Natural Language Processing (NLP) involves resolving these ambiguities. Early work in NLP focused on a small number of well-formed corpora in a small number of languages, but significant advances have been made in recent years by using large and diverse corpora from a wide range of sources, including a vast and ever-growing supply of dynamically generated text from the Internet. This explosion in corpus size and variety has necessitated techniques for automatically harvesting and preparing text corpora for NLP tasks.

In this chapter, we discuss the challenges posed by **text preprocessing**, the task of converting a raw text file, essentially a sequence of digital bits, into a well-defined sequence of linguistically meaningful units: at the lowest level characters representing the individual graphemes in a language's written system, words consisting of one or more characters, and sentences consisting of one or more words. Text preprocessing is an essential part of any NLP system, since the characters, words, and sentences identified at this stage are the fundamental units passed to all further processing stages, from analysis and tagging components, such as morphological analyzers and part-of-speech taggers, through applications, such as information retrieval and machine translation systems.

Text preprocessing can be divided into two stages: document triage and text segmentation. **Document**

triage is the process of converting a set of digital files into well-defined text documents. For early corpora, this was a slow, manual process, and these early corpora were rarely more than a few million words.

In contrast, current corpora harvested from the Internet can encompass billions of words *each day*, which requires a fully automated document triage process. This process can involve several steps, depending on the origin of the files being processed. First, in order for any natural language document to be machine readable, its characters must be represented in a character encoding, in which one or more bytes in a file maps to a known character. **Character encoding identification** determines the character encoding (or encodings) for any file and optionally converts between encodings. Second, in order to know what language-specific algorithms to apply to a document, **language identification** determines the natural language for a document; this step is closely linked to, but not uniquely determined by, the character encoding. Finally, **text sectioning** identifies the actual content within a file while discarding undesirable elements, such as images, tables, headers, links, and HTML formatting. The output of the document triage stage is a well-defined text corpus, organized by language, suitable for text segmentation and further analysis.

Text segmentation is the process of converting a well-defined text corpus into its component words and sentences. **Word segmentation** breaks up the sequence of characters in a text by locating the **word boundaries**, the points where one word ends and another begins. For computational linguistics purposes, the words thus identified are frequently referred to as **tokens**, and word segmentation is also known as **tokenization**. **Text normalization** is a related step that involves merging different written forms of a token into a canonical normalized form; for example, a document may contain the equivalent tokens “Mr.”, “Mr”, “mister”, and “Mister” that would all be normalized to a single form.

Sentence segmentation is the process of determining the longer processing units consisting of one or more words. This task involves identifying **sentence boundaries** between words in different sentences. Since most written languages have punctuation marks that occur at sentence boundaries, sentence segmentation is frequently referred to as **sentence boundary detection**, **sentence boundary disambiguation**, or **sentence boundary recognition**. All these terms refer to the same task: determining how a text should be divided into sentences for further processing.

In practice, sentence and word segmentation cannot be performed successfully independent from one another. For example, an essential subtask in both word and sentence segmentation for most European languages is identifying abbreviations, because a period can be used to mark an abbreviation as well as to mark the end of a sentence. In the case of a period marking an abbreviation, the period is usually considered a part of the abbreviation token, whereas a period at the end of a sentence is usually considered a token in and of itself. In the case of an abbreviation at the end of a sentence, the period marks *both* the abbreviation and the sentence boundary.

This chapter provides an introduction to text preprocessing in a variety of languages and writing systems. We begin in Section 2.2 with a discussion of the challenges posed by text preprocessing, and emphasize the document triage issues that must be considered before implementing a tokenization or sentence segmentation algorithm. The section describes the dependencies on the language being processed and the character set in which the language is encoded. It also discusses the dependency on the application that uses the output of the segmentation and the dependency on the characteristics of the specific corpus being processed.

In Section 2.3, we introduce some common techniques currently used for tokenization. The first part of the section focuses on issues that arise in tokenizing and normalizing languages in which words are separated by whitespace. The second part of the section discusses tokenization techniques in languages where no such whitespace word boundaries exist. In Section 2.4, we discuss the problem of sentence segmentation and introduce some common techniques currently used to identify sentence boundaries in texts.

2.2 Challenges of Text Preprocessing

There are many issues that arise in text preprocessing that need to be addressed when designing NLP

systems, and many can be addressed as part of document triage in preparing a corpus for analysis.

The type of writing system used for a language is the most important factor for determining the best approach to text preprocessing. Writing systems can be **logographic**, where a large number (often thousands) of individual symbols represent words. In contrast, writing systems can be **syllabic**, in which individual symbols represent syllables, or **alphabetic**, in which individual symbols (more or less) represent sounds; unlike logographic systems, syllabic and alphabetic systems typically have fewer than 100 symbols. According to Comrie et al. (1996), the majority of all written languages use an alphabetic or syllabic system. However, in practice, no modern writing system employs symbols of only one kind, so no natural language writing system can be classified as purely logographic, syllabic, or alphabetic. Even English, with its relatively simple writing system based on the Roman alphabet, utilizes logographic symbols including Arabic numerals (0–9), currency symbols (\$, £), and other symbols (% , & , #). English is nevertheless predominately alphabetic, and most other writing systems are comprised of symbols which are mainly of one type.

In this section, we discuss the essential document triage steps, and we emphasize the main types of dependencies that must be addressed in developing algorithms for text segmentation: **character-set dependence** (Section 2.2.1), **language dependence** (Section 2.2.2), **corpus dependence** (Section 2.2.3), and **application dependence** (Section 2.2.4).

2.2.1 Character-Set Dependence

At its lowest level, a computer-based text or document is merely a sequence of digital bits in a file. The first essential task is to interpret these bits as characters of a writing system of a natural language.

2.2.1.1 About Character Sets

Historically, interpreting digital text files was trivial, since nearly all texts were encoded in the **7-bit character set** ASCII, which allowed only 128 (2^7) characters and included only the Roman (or Latin) alphabet and essential characters for writing English. This limitation required the “asciification” or “romanization” of many texts, in which ASCII equivalents were defined for characters not defined in the character set. An example of this asciification is the adaptation of many European languages containing umlauts and accents, in which the umlauts are replaced by a double quotation mark or the letter ‘e’ and accents are denoted by a single quotation mark or even a number code. In this system, the German word *über* would be written as *u"ber* or *ueber*, and the French word *déjà* would be written as *de'ja* or *de1ja2*. Languages that do not use the roman alphabet, such as Russian and Arabic, required much more elaborate romanization systems, usually based on a phonetic mapping of the source characters to the roman characters. The Pinyin transliteration of Chinese writing is another example of asciification of a more complex writing system. These adaptations are still common due to the widespread familiarity with the roman characters; in addition, some computer applications are still limited to this 7-bit encoding.

Eight-bit character sets can encode 256 (2^8) characters using a single 8-bit byte, but most of these

8-bit sets reserve the first 128 characters for the original ASCII characters. Eight-bit encodings exist for all common alphabetic and some syllabic writing systems; for example, the ISO-8859 series of 10+ character sets contains encoding definitions for most European characters, including separate ISO-8859 sets for the Cyrillic and Greek alphabets. However, since all 8-bit character sets are limited to exactly the same 256 byte codes (decimal 0–255), this results in a large number of overlapping character sets for encoding characters in different languages.

Writing systems with larger character sets, such as those of written Chinese and Japanese, which have several thousand distinct characters, require multiple bytes to encode a single character. A **two-byte character set** can represent 65,536 (2^{16}) distinct characters, since 2 bytes contain 16 bits. Determining individual characters in two-byte character sets involves grouping pairs of bytes representing a single character. This process can be complicated by the tokenization equivalent of code-switching, in which

characters from many different writing systems occur within the same text. It is very common in digital texts to encounter multiple writing systems and thus multiple encodings, or as discussed previously,

character encodings that include other encodings as subsets. In Chinese and Japanese texts, single-byte letters, spaces, punctuation marks (e.g., periods, quotation marks, and parentheses), and Arabic numerals (0–9) are commonly interspersed with 2-byte Chinese and Japanese characters. Such texts also frequently contain ASCII headers. Multiple encodings also exist for these character sets; for example, the Chinese character set is represented in two widely used encodings, Big-5 for the complex-form (traditional) character set and GB for the simple-form (simplified) set, with several minor variants of these sets also commonly found.

The **Unicode 5.0 standard** (Unicode Consortium 2006) seeks to eliminate this character set ambiguity by specifying a Universal Character Set that includes over 100,000 distinct coded characters derived from over 75 supported scripts representing all the writing systems commonly used today. The Unicode standard is most commonly implemented in the **UTF-8 variable-length character encoding**, in which each character is represented by a 1 to 4 byte encoding. In the UTF-8 encoding, ASCII characters require 1 byte, most other characters included in ISO-8859 character encodings and other alphabetic systems require 2 bytes, and all other characters, including Chinese, Japanese, and Korean, require 3 bytes (and very rarely 4 bytes). The Unicode standard and its implementation in UTF-8 allow for the encoding of all supported characters with no overlap or confusion between conflicting byte ranges, and it is rapidly replacing older character encoding sets for multilingual applications.

2.2.1.2 Character Encoding Identification and Its Impact on Tokenization

Despite the growing use of Unicode, the fact that the same range of numeric values can represent different characters in different encodings can be a problem for tokenization. For example, English or Spanish are both normally stored in the common 8-bit encoding Latin-1 (or ISO-8859-1). An English or Spanish tokenizer would need to be aware that bytes in the (decimal) range 161–191 in Latin-1 represent punctuation marks and other symbols (such as ‘í’, ‘¿’, ‘£’, and ‘Š’). Tokenization rules would then be required to handle each symbol (and thus its byte code) appropriately for that language. However, this same byte range in UTF-8 represents the second (or third or fourth) byte of a multi-byte sequence and is meaningless by itself; an English or Spanish tokenizer for UTF-8 would thus need to model multi-byte character sequences explicitly. Furthermore, the same byte range in ISO-8859-5, a common Russian encoding, contains Cyrillic characters; in KOI8-R, another Russian encoding, the range contains a *different set* of Cyrillic characters. Tokenizers thus must be targeted to a specific language in a specific encoding.

Tokenization is unavoidably linked to the underlying character encoding of the text being processed, and character encoding identification is an essential first step. While the header of a digital document may contain information regarding its character encoding, this information is not always present or even reliable, in which case the encoding must be determined automatically.

A character encoding identification algorithm must first explicitly model the known encoding systems, in order to know in what byte ranges to look for valid characters as well as which byte ranges are unlikely to appear frequently in that encoding. The algorithm then analyzes the bytes in a file to construct a profile of which byte ranges are represented in the file. Next, the algorithm compares the patterns of bytes found in the file to the expected byte ranges from the known encodings and decides which encoding best fits the data.

Russian encodings provide a good example of the different byte ranges encountered for a given language. In the ISO-8859-5 encoding for Russian texts, the capital Cyrillic letters are in the (hexadecimal) range B0–CF (and are listed in the traditional Cyrillic alphabetical order); the lowercase letters are in the range D0–EF. In contrast, in the KOI8-R encoding, the capital letters are E0–FF (and are listed in pseudo-Roman order); the lowercase letters are C0–DF. In Unicode, Cyrillic characters require two bytes, and the capital letters are in the range 0410 (the byte 04 followed by the byte 10) through 042F; the lowercase letters are in the range 0430–045F. A character encoding identification algorithm seeking to determine the encoding of a given Russian text would examine the bytes contained in the file to determine the byte ranges present.

The hex byte 04 is a rare control character in ISO-8859-5 and in KOI8-R but would comprise nearly half

the bytes in a Unicode Russian file. Similarly, a file in ISO-8859-5 would likely contain many bytes in the range B0-BF but few in F0-FF, while a file in KOI8-R would contain few in B0-BF and many in F0-FF. Using these simple heuristics to analyze the byte distribution in a file should allow for straightforward encoding identification for Russian texts.

Note that, due to the overlap between existing character encodings, even with a high-quality character encoding classifier, it may be impossible to determine the character encoding. For example, since most character encodings reserve the first 128 characters for the ASCII characters, a document that contains only these 128 characters could be any of the ISO-8859 encodings or even UTF-8.

2.2.2 Language Dependence

2.2.2.1 Impact of Writing System on Text Segmentation

In addition to the variety of symbol types (logographic, syllabic, or alphabetic) used in writing systems, there is a range of orthographic conventions used in written languages to denote the boundaries between linguistic units such as syllables, words, or sentences. In many written Amharic texts, for example, both word and sentence boundaries are explicitly marked, while in written Thai texts neither is marked. In the latter case, where no boundaries are explicitly indicated in the written language, written Thai is similar to spoken language, where there are no explicit boundaries and few cues to indicate segments at any level. Between the two extremes are languages that mark boundaries to different degrees. English employs whitespace between most words and punctuation marks at sentence boundaries, but neither feature is sufficient to segment the text completely and unambiguously. Tibetan and Vietnamese both explicitly mark syllable boundaries, either through layout or by punctuation, but neither marks word boundaries. Written Chinese and Japanese have adopted punctuation marks for sentence boundaries, but neither denotes word boundaries.

In this chapter, we provide general techniques applicable to a variety of different writing systems. Since many segmentation issues are language-specific, we will also highlight the challenges faced by robust, broad-coverage tokenization efforts. For a very thorough description of the various writing systems employed to represent natural languages, including detailed examples of all languages and features discussed in this chapter, we recommend Daniels and Bright (1996).

2.2.2.2 Language Identification

The wide range of writing systems used by the languages of the world result in language-specific as well as orthography-specific features that must be taken into account for successful text segmentation. An important step in the document triage stage is thus to identify the language of each document or document section, since some documents are multilingual at the section level or even paragraph level.

For languages with a unique alphabet not used by any other languages, such as Greek or Hebrew, language identification is determined by character set identification. Similarly, character set identification can be used to narrow the task of language identification to a smaller number of languages that all share many characters, such as Arabic vs. Persian, Russian vs. Ukrainian, or Norwegian vs. Swedish. The byte range distribution used to determine character set identification can further be used to identify bytes, and thus characters, that are predominant in one of the remaining candidate languages, if the languages do not share exactly the same characters. For example, while Arabic and Persian both use the Arabic alphabet, the Persian language uses several supplemental characters that do not appear in Arabic. For more difficult cases, such as European languages that use exactly the same character set but with different frequencies, final identification can be performed by training models of byte/character distributions in each of the languages. A basic but very effective algorithm for this would sort the bytes in a file by frequency count

and use the sorted list as a signature vector for comparison via an n-gram or vector distance model.

2.2.3 Corpus Dependence

Early NLP systems rarely addressed the problem of robustness, and they normally could process only well-formed input conforming to their hand-built grammars. The increasing availability of large corpora in multiple languages that encompass a wide range of data types (e.g., newswire texts, email messages, closed captioning data, Internet news pages, and weblogs) has required the development of robust NLP approaches, as these corpora frequently contain misspellings, erratic punctuation and spacing, and other irregular features. It has become increasingly clear that algorithms which rely on input texts to be well-formed are much less successful on these different types of texts.

Similarly, algorithms that expect a corpus to follow a set of conventions for a written language are frequently not robust enough to handle a variety of corpora, especially those harvested from the Internet. It is notoriously difficult to prescribe rules governing the use of a written language; it is even more difficult to get people to “follow the rules.” This is in large part due to the nature of written language, in which the conventions are not always in line with actual usage and are subject to frequent change. So while punctuation roughly corresponds to the use of suprasegmental features in spoken language, reliance on well-formed sentences delimited by predictable punctuation can be very problematic. In many corpora, traditional prescriptive rules are commonly ignored. This fact is particularly important to our discussion of both word and sentence segmentation, which to a large degree depends on the regularity of spacing and punctuation. Most existing segmentation algorithms for natural languages are both language-specific and corpus-dependent, developed to handle the predictable ambiguities in a well-formed text. Depending on the origin and purpose of a text, capitalization and punctuation rules may be followed very closely (as in most works of literature), erratically (as in various newspaper texts), or not at all (as in email messages and personal Web pages). Corpora automatically harvested from the Internet can be especially ill-formed, such as Example (1), an actual posting to a Usenet newsgroup, which shows the erratic use of capitalization and punctuation, “creative” spelling, and domain-specific terminology inherent in such texts.

(1) *ive just loaded pcl onto my akcl. when i do an 'in- package' to load pcl, ill get the prompt but im not able to use functions like defclass, etc... is there womething basic im missing or am i just left hanging, twisting in the breeze?*

Many digital text files, such as those harvested from the Internet, contain large regions of text that are undesirable for the NLP application processing the file. For example, Web pages can contain headers, images, advertisements, site navigation links, browser scripts, search engine optimization terms, and other markup, little of which is considered actual *content*. Robust text segmentation algorithms designed for use with such corpora must therefore have the capability to handle the range of irregularities, which distinguish these texts from well-formed corpora. A key task in the document triage stage for such files is text sectioning, in which extraneous text is removed. The sectioning and cleaning of Web pages has recently become the focus of Cleaneval, “a shared task and competitive evaluation on the topic of cleaning arbitrary Web pages, with the goal of preparing Web data for use as a corpus for linguistic and language technology research and development.” (Baroni et al. 2008)

2.2.4 Application Dependence

Although word and sentence segmentation are necessary, in reality, there is no absolute definition for what constitutes a word or a sentence. Both are relatively arbitrary distinctions that vary greatly across written languages. However, for the purposes of computational linguistics we need to define exactly what we need for further processing; in most cases, the language and task at hand determine the necessary conventions. For example, the English words *I am* are frequently contracted to *I'm*, and a tokenizer frequently expands the contraction to recover the essential grammatical features of the pronoun and the verb. A tokenizer that does not expand this contraction to the component words would pass the single token *I'm* to later

processing stages. Unless these processors, which may include morphological analyzers, part-of-speech

taggers, lexical lookup routines, or parsers, are aware of both the contracted and uncontracted forms, the token may be treated as an unknown word.

Another example of the dependence of tokenization output on later processing stages is the treatment of the English possessive 's in various tagged corpora.^{*} In the Brown corpus (Francis and Kucera 1982), the word *governor's* is considered one token and is tagged as a possessive noun. In the Susanne corpus (Sampson 1995), on the other hand, the same word is treated as two tokens, *governor* and 's, tagged singular noun and possessive, respectively.

Examples such as the above are usually addressed during tokenization by normalizing the text to meet the requirements of the applications. For example, language modeling for automatic speech recognition requires that the tokens be represented in a form similar to how they are spoken (and thus input to the speech recognizer). For example, the written token \$300 would be spoken as "three hundred dollars," and the text normalization would convert the original to the desired three tokens. Other applications may require that this and all other monetary amounts be converted to a single token such as "MONETARY_TOKEN."

In languages such as Chinese, which do not contain white space between any words, a wide range of word segmentation conventions are currently in use. Different segmentation standards have a significant impact on applications such as information retrieval and text-to-speech synthesis, as discussed in Wu (2003). Task-oriented Chinese segmentation has received a great deal of attention in the MT community (Chang et al. 2008; Ma and Way 2009; Zhang et al. 2008).

The tasks of word and sentence segmentation overlap with the techniques discussed in many other chapters in this handbook, in particular the chapters on Lexical Analysis, Corpus Creation, and Multiword Expressions, as well as practical applications discussed in other chapters.

2.3 Tokenization

Section 2.2 discussed the many challenges inherent in segmenting freely occurring text. In this section, we focus on the specific technical issues that arise in tokenization.

Tokenization is well-established and well-understood for artificial languages such as programming languages.[†] However, such artificial languages can be strictly defined to eliminate lexical and structural ambiguities; we do not have this luxury with natural languages, in which the same character can serve many different purposes and in which the syntax is not strictly defined. Many factors can affect the difficulty of tokenizing a particular natural language. One fundamental difference exists between tokenization approaches for **space-delimited** languages and approaches for **unsegmented** languages. In space-delimited languages, such as most European languages, some word boundaries are indicated by the insertion of whitespace. The character sequences delimited are not necessarily the tokens required for further processing, due both to the ambiguous nature of the writing systems and to the range of tokenization conventions required by different applications. In unsegmented languages, such as Chinese and Thai, words are written in succession with no indication of word boundaries. The tokenization of unsegmented languages therefore requires additional lexical and morphological information.

In both unsegmented and space-delimited languages, the specific challenges posed by tokenization are largely dependent on both the writing system (logographic, syllabic, or alphabetic, as discussed in Section 2.2.2) and the typographical structure of the words. There are three main categories into which word structures can be placed,[‡] and each category exists in both unsegmented and space-delimited writing systems. The morphology of words in a language can be **isolating**, where words do not divide into smaller units; **agglutinating** (or **agglutinative**), where words divide into smaller units (morphemes) with clear

^{*} This example is taken from Grefenstette and Tapanainen (1994).

[†] For a thorough introduction to the basic techniques of tokenization in programming languages, see Aho et al. (1986).

[‡] This classification comes from Comrie et al. (1996) and Crystal (1987).

boundaries between the morphemes; or **inflectional**, where the boundaries between morphemes are not clear and where the component morphemes can express more than one grammatical meaning. While individual languages show tendencies toward one specific type (e.g., Mandarin Chinese is predominantly isolating, Japanese is strongly agglutinative, and Latin is largely inflectional), most languages exhibit traces of all three. A fourth typological classification frequently studied by linguists, **polysynthetic**, can be considered an extreme case of agglutinative, where several morphemes are put together to form complex words that can function as a whole sentence. Chukchi and Inuktitut are examples of polysynthetic languages, and some research in machine translation has focused on a Nunavut Hansards parallel corpus of Inuktitut and English (Martin et al. 2003).

Since the techniques used in tokenizing space-delimited languages are very different from those used in tokenizing unsegmented languages, we discuss the techniques separately in Sections 2.3.1 and 2.3.2, respectively.

2.3.1 Tokenization in Space-Delimited Languages

In many alphabetic writing systems, including those that use the Latin alphabet, words are separated by whitespace. Yet even in a well-formed corpus of sentences, there are many issues to resolve in tokenization. Most tokenization ambiguity exists among uses of punctuation marks, such as periods, commas, quotation marks, apostrophes, and hyphens, since the same punctuation mark can serve many different functions in a single sentence, let alone a single text. Consider example sentence (3) from the *Wall Street Journal* (1988).

(3) *Clairson International Corp. said it expects to report a net loss for its second quarter ended March 26 and doesn't expect to meet analysts' profit estimates of \$3.9 to \$4 million, or 76 cents a share to 79 cents a share, for its year ending Sept. 24.*

This sentence has several items of interest that are common for Latinate, alphabetic, space-delimited languages. First, it uses periods in three different ways: within numbers as a decimal point (\$3.9), to mark abbreviations (*Corp.* and *Sept.*), and to mark the end of the sentence, in which case the period following the number 24 is not a decimal point. The sentence uses apostrophes in two ways: to mark the genitive case (where the apostrophe denotes possession) in *analysts'* and to show contractions (places where letters have been left out of words) in *doesn't*. The tokenizer must thus be aware of the uses of punctuation marks and be able to determine when a punctuation mark is part of another token and when it is a separate token.

In addition to resolving these cases, we must make tokenization decisions about a phrase such as *76 cents a share*, which on the surface consists of four tokens. However, when used adjectivally such as in the phrase *a 76-cents-a-share dividend*, it is normally hyphenated and appears as one. The semantic content is the same despite the orthographic differences, so it makes sense to treat the two identically, as the same number of tokens. Similarly, we must decide whether to treat the phrase *\$3.9 to \$4 million* differently than if it had been written as *3.9 to 4 million dollars* or *\$3,900,000 to \$4,000,000*. Note also that the semantics of numbers can be dependent on both the genre and the application; in scientific literature, for example, the numbers 3.9, 3.90, and 3.900 have different significant digits and are not semantically equivalent. We discuss these ambiguities and other issues in the following sections.

A logical initial tokenization of a space-delimited language would be to consider as a separate token any sequence of characters preceded and followed by space. This successfully tokenizes words that are a sequence of alphabetic characters, but does not take into account punctuation characters. In many cases, characters such as commas, semicolons, and periods should be treated as separate tokens, although they are not preceded by whitespace (such as the case with the comma after *\$4 million* in Example (3)). Additionally, many texts contain certain classes of character sequences which should be filtered out before actual tokenization; these include existing markup and headers (including HTML markup), extra

whitespace, and extraneous control characters.

2.3.1.1 Tokenizing Punctuation

While punctuation characters are usually treated as separate tokens, there are many cases when they should be “attached” to another token. The specific cases vary from one language to the next, and the specific treatment of the punctuation characters needs to be enumerated within the tokenizer for each language. In this section, we give examples of English tokenization.

Abbreviations are used in written language to denote the shortened form of a word. In many cases, abbreviations are written as a sequence of characters terminated with a period. When an abbreviation occurs at the end of a sentence, a single period marks both the abbreviation and the sentence boundary. For this reason, recognizing abbreviations is essential for both tokenization and sentence segmentation. Compiling a list of abbreviations can help in recognizing them, but abbreviations are productive, and it is not possible to compile an exhaustive list of all abbreviations in any language. Additionally, many abbreviations can also occur as words elsewhere in a text (e.g., the word *Mass* is also the abbreviation for *Massachusetts*). An abbreviation can also represent several different words, as is the case for *St.* which can stand for *Saint*, *Street*, or *State*. However, as *Saint* it is less likely to occur at a sentence boundary than as *Street*, or *State*. Examples (4) and (5) from the *Wall Street Journal* (1991 and 1987 respectively) demonstrate the difficulties produced by such ambiguous cases, where the same abbreviation can represent different words and can occur both within and at the end of a sentence.

(4) *The contemporary viewer may simply ogle the vast wooded vistas rising up from the Saguenay River and Lac St. Jean, standing in for the St. Lawrence River.*

(5) *The firm said it plans to sublease its current headquarters at 55 Water St. A spokesman declined to elaborate.*

Recognizing an abbreviation is thus not sufficient for complete tokenization, and the appropriate definition for an abbreviation can be ambiguous, as discussed in Park and Byrd (2001). We address abbreviations at sentence boundaries fully in Section 2.4.2.

Quotation marks and apostrophes (‘’’) are a major source of tokenization ambiguity. In most cases, single and double quotes indicate a quoted passage, and the extent of the tokenization decision is to determine whether they open or close the passage. In many character sets, single quote and apostrophe are the same character, and it is therefore not always possible to immediately determine if the single quotation mark closes a quoted passage, or serves another purpose as an apostrophe. In addition, as discussed in Section 2.2.1, quotation marks are also commonly used when “romanizing” writing systems, in which umlauts are replaced by a double quotation mark and accents are denoted by a single quotation mark or an apostrophe.

The apostrophe is a very ambiguous character. In English, the main uses of apostrophes are to mark the genitive form of a noun, to mark contractions, and to mark certain plural forms. In the genitive case, some applications require a separate token while some require a single token, as discussed in Section 2.2.4. How to treat the genitive case is important, since in other languages, the possessive form of a word is not marked with an apostrophe and cannot be as readily recognized. In German, for example, the possessive form of a noun is usually formed by adding the letter *s* to the word, without an apostrophe, as in *Peters Kopf* (*Peter’s head*). However, in modern (informal) usage in German, *Peter’s Kopf* would also be common; the apostrophe is also frequently omitted in modern (informal) English such that *Peters head* is a possible construction. Furthermore, in English, *’s* can serve as a contraction for the verb *is*, as in *he’s*, *it’s*, *she’s*, and *Peter’s head and shoulders above the rest*. It also occurs in the plural form of some words, such as *I.D.’s* or *1980’s*, although the apostrophe is also frequently omitted from such plurals. The tokenization decision in these cases is context dependent and is closely tied to syntactic analysis.

In the case of apostrophe as contraction, tokenization may require the expansion of the word to eliminate the apostrophe, but the cases where this is necessary are very language-dependent. The English contraction *I’m* could be tokenized as the two words *I am*, and *we’ve* could become *we have*. Written French

contains a completely different set of contractions, including contracted articles (*l'homme, c'était*), as well

as contracted pronouns (*j'ai, je l'ai*) and other forms such as *n'y, qu'ils, d'ailleurs*, and *aujourd'hui*. Clearly, recognizing the contractions to expand requires knowledge of the language, and the specific contractions to expand, as well as the expanded forms, must be enumerated. All other word-internal apostrophes are treated as a part of the token and not expanded, which allows the proper tokenization of multiply-contracted words such as *fo'c's'le* (forecastle) and *Pudd'n'head* (Puddinghead) as single words. In addition, since contractions are not always demarcated with apostrophes, as in the French *du*, which is a contraction of *de le*, or the Spanish *del*, contraction of *de el*, other words to expand must also be listed in the tokenizer.

2.3.1.2 Multi-Part Words

To different degrees, many written languages contain space-delimited words composed of multiple units, each expressing a particular grammatical meaning. For example, the single Turkish word *çöplüklerim-izdekilerdenmiydi* means “was it from those that were in our garbage cans?”* This type of construction is particularly common in strongly agglutinative languages such as Swahili, Quechua, and most Altaic languages. It is also common in languages such as German, where noun–noun (*Lebensversicherung*, life insurance), adverb–noun (*Nichtraucher*, nonsmoker), and preposition–noun (*Nachkriegszeit*, postwar period) compounding are all possible. In fact, though it is not an agglutinative language, German compounding can be quite complex, as in *Feuerundlebensversicherung* (fire and life insurance) or *Kundenzufriedenheitsabfragen* (customer satisfaction survey).

To some extent, agglutinating constructions are present in nearly all languages, though this compounding can be marked by hyphenation, in which the use of hyphens can create a single word with multiple grammatical parts. In English, it is commonly used to create single-token words like *end-of-line* as well as multi-token words like *Boston-based*. As with the apostrophe, the use of the hyphen is not uniform; for example, hyphen usage varies greatly between British and American English, as well as between different languages. However, as with the case of apostrophes as contractions, many common language-specific uses of hyphens can be enumerated in the tokenizer.

Many languages use the hyphen to create essential grammatical structures. In French, for example, hyphenated compounds such as *va-t-il* (will it?), *c'est-à-dire* (that is to say), and *celui-ci* (it) need to be expanded during tokenization, in order to recover necessary grammatical features of the sentence. In these cases, the tokenizer needs to contain an enumerated list of structures to be expanded, as with the contractions discussed above.

Another tokenization difficulty involving hyphens stems from the practice, common in traditional typesetting, of using hyphens at the ends of lines to break a word too long to include on one line. Such end-of-line hyphens can thus occur within words that are not normally hyphenated. Removing these hyphens is necessary during tokenization, yet it is difficult to distinguish between such incidental hyphenation and cases where naturally hyphenated words happen to occur at a line break. In an attempt to dehyphenate the artificial cases, it is possible to incorrectly remove necessary hyphens. Grefenstette and Tapanainen (1994) found that nearly 5% of the end-of-line hyphens in an English corpus were word-internal hyphens, which happened to also occur as end-of-line hyphens.

In tokenizing multi-part words, such as hyphenated or agglutinative words, whitespace does not provide much useful information to further processing stages. In such cases, the problem of tokenization is very closely related both to tokenization in unsegmented languages, discussed in Section 2.3.2, and to morphological analysis, discussed in Chapter 3 of this handbook.

2.3.1.3 Multiword Expressions

Spacing conventions in written languages do not always correspond to the desired tokenization for NLP applications, and the resulting multiword expressions are an important consideration in the tokenization stage. A later chapter of this handbook addresses Multiword Expressions in full detail, so we touch briefly in this section on some of the tokenization issues raised by multiword expressions.

* This example is from Hankamer (1986).

For example, the three-word English expression *in spite of* is, for all intents and purposes, equivalent to the single word *despite*, and both could be treated as a single token. Similarly, many common English expressions, such as *au pair*, *de facto*, and *joie de vivre*, consist of foreign loan words that can be treated as a single token.

Multiword numerical expressions are also commonly identified in the tokenization stage. Numbers are ubiquitous in all types of texts in every language, but their representation in the text can vary greatly. For most applications, sequences of digits and certain types of numerical expressions, such as dates and times, money expressions, and percents, can be treated as a single token. Several examples of such phrases can be seen in Example (3) above: *March 26*, *\$3.9 to \$4 million*, and *Sept. 24* could each be treated as a single token. Similarly, phrases such as *76 cents a share* and *\$3-a-share* convey roughly the same meaning, despite the difference in hyphenation, and the tokenizer should normalize the two phrases to the same number of tokens (either one or four). Tokenizing numeric expressions requires the knowledge of the syntax of such expressions, since numerical expressions are written differently in different languages. Even within a language or in languages as similar as English and French, major differences exist in the syntax of numeric expressions, in addition to the obvious vocabulary differences. For example, the English date *November 18, 1989* could alternately appear in English texts as any number of variations, such as *Nov. 18, 1989*, *18 November 1989*, *11/18/89* or *18/11/89*. These examples underscore the importance of text normalization during the tokenization process, such that dates, times, monetary expressions, and all other numeric phrases can be converted into a form that is consistent with the processing required by the NLP application.

Closely related to hyphenation, the treatment of multiword expressions is highly language-dependent and application-dependent, but can easily be handled in the tokenization stage if necessary. We need to be careful, however, when combining words into a single token. The phrase *no one*, along with *noone* and *no-one*, is a commonly encountered English equivalent for *nobody*, and should normally be treated as a single token. However, in a context such as *No one man can do it alone*, it needs to be treated as two words. The same is true of the two-word phrase *can not*, which is not always equivalent to the single word *cannot* or the contraction *can't*.^{*} In such cases, it is safer to allow a later process (such as a parser) to make the decision.

2.3.2 Tokenization in Unsegmented Languages

The nature of the tokenization task in unsegmented languages like Chinese, Japanese, and Thai is fundamentally different from tokenization in space-delimited languages like English. The lack of any spaces between words necessitates a more informed approach than simple lexical analysis. The specific approach to word segmentation for a particular unsegmented language is further limited by the writing system and orthography of the language, and a single general approach has not been developed. In Section 2.3.2.1, we describe some algorithms, which have been applied to the problem to obtain an initial approximation for a variety of languages. In Sections 2.3.2.2 and 2.3.2.3, we give details of some successful approaches to Chinese and Japanese segmentation, and in Section 2.3.2.4, we describe some approaches, which have been applied to languages with unsegmented alphabetic or syllabic writing systems.

2.3.2.1 Common Approaches

An extensive word list combined with an informed segmentation algorithm can help to achieve a certain degree of accuracy in word segmentation, but the greatest barrier to accurate word segmentation is in recognizing unknown (or out-of-vocabulary) words, words not in the lexicon of the segmenter. This problem is dependent both on the source of the lexicon as well as the correspondence (in vocabulary) between the text in question and the lexicon; for example, Wu and Fung (1994) reported that segmentation

* For example, consider the following sentence: "Why is my soda can not where I left it?"

accuracy in Chinese is significantly higher when the lexicon is constructed using the same type of corpus as the corpus on which it is tested.

Another obstacle to high-accuracy word segmentation is the fact that there are no widely accepted guidelines as to what constitutes a word, and there is therefore no agreement on how to “correctly” segment a text in an unsegmented language. Native speakers of a language do not always agree about the “correct” segmentation, and the same text could be segmented into several very different (and equally correct) sets of words by different native speakers. A simple example from English would be the hyphenated phrase *Boston-based*. If asked to “segment” this phrase into words, some native English speakers might say *Boston-based* is a single word and some might say *Boston* and *based* are two separate words; in this latter case there might also be disagreement about whether the hyphen “belongs” to one of the two words (and to which one) or whether it is a “word” by itself. Disagreement by native speakers of Chinese is much more prevalent; in fact, Sproat et al. (1996) give empirical results showing that native speakers of Chinese agree on the correct segmentation in fewer than 70% of the cases. Such ambiguity in the definition of what constitutes a word makes it difficult to evaluate segmentation algorithms that follow different conventions, since it is nearly impossible to construct a “gold standard” against which to directly compare results.

A simple word segmentation algorithm consists of considering each character to be a distinct word. This is practical for Chinese because the average word length is very short (usually between one and two characters, depending on the corpus*) and actual words can be recognized with this algorithm. Although it does not assist in tasks such as parsing, part-of-speech tagging, or text-to-speech systems (see Sproat et al. 1996), the character-as-word segmentation algorithm is very common in Chinese information retrieval, a task in which the words in a text play a major role in indexing and where incorrect segmentation can hurt system performance.

A very common approach to word segmentation is to use a variation of the *maximum matching algorithm*, frequently referred to as the *greedy algorithm*. The greedy algorithm starts at the first character in a text and, using a word list for the language being segmented, attempts to find the longest word in the list starting with that character. If a word is found, the maximum-matching algorithm marks a boundary at the end of the longest word, then begins the same longest match search starting at the character following the match. If no match is found in the word list, the greedy algorithm simply segments that character as a word (as in the character-as-word algorithm above) and begins the search starting at the next character. A variation of the greedy algorithm segments a sequence of unmatched characters as a single word; this variant is more likely to be successful in writing systems with longer average word lengths. In this manner, an initial segmentation can be obtained that is more informed than a simple character-as-word approach. The success of this algorithm is largely dependent on the word list.

As a demonstration of the application of the character-as-word and greedy algorithms, consider an example of artificially “desegmented” English, in which all the white space has been removed. The desegmented version of the phrase *the table down there* would thus be *thetabledownthere*. Applying the character-as-word algorithm would result in the useless sequence of tokens *t h e t a b l e d o w n t h e r e*, which is why this algorithm only makes sense for languages with short average word length, such as Chinese. Applying the greedy algorithm with a “perfect” word list containing all known English words would first identify the word *theta*, since that is the longest sequence of letters starting at the initial *t*, which forms an actual word. Starting at the *b* following *theta*, the algorithm would then identify *bled* as the maximum match. Continuing in this manner, *thetabledownthere* would be segmented by the greedy algorithm as *theta bled own there*.

A variant of the maximum matching algorithm is the *reverse maximum matching algorithm*, in which the matching proceeds from the end of the string of characters, rather than the beginning. In the example above, *thetabledownthere* would be correctly segmented as *the table down there* by the reverse maximum matching algorithm. Greedy matching from the beginning and the end of the string of characters enables an

* As many as 95% of Chinese words consist of one or two characters, according to Fung and Wu (1994).

algorithm such as *forward-backward matching*, in which the results are compared and the segmentation optimized based on the two results. In addition to simple greedy matching, it is possible to encode language-specific heuristics to refine the matching as it progresses.

2.3.2.2 Chinese Segmentation

The Chinese writing system consists of several thousand characters known as *Hanzi*, with a word consisting of one or more characters. In this section, we provide a few examples of previous approaches to Chinese word segmentation, but a detailed treatment is beyond the scope of this chapter. Much of our summary is taken from Sproat et al. (1996) and Sproat and Shih (2001). For a comprehensive summary of early work in Chinese segmentation, we also recommend Wu and Tseng (1993).

Most previous work in Chinese segmentation falls into one of the three categories: statistical approaches, lexical rule-based approaches, and hybrid approaches that use both statistical and lexical information. Statistical approaches use data such as the mutual information between characters, compiled from a training corpus, to determine which characters are most likely to form words. Lexical approaches use manually encoded features about the language, such as syntactic and semantic information, common phrasal structures, and morphological rules, in order to refine the segmentation. The hybrid approaches combine information from both statistical and lexical sources.

Sproat et al. (1996) describe such a hybrid approach that uses a weighted finite-state transducer to identify both dictionary entries as well as unknown words derived by productive lexical processes. Palmer (1997) also describes a hybrid statistical-lexical approach in which the segmentation is incrementally improved by a trainable sequence of transformation rules; Hockenmaier and Brew (1998) describe a similar approach. Teahan et al. (2000) describe a novel approach based on adaptive language models similar to those used in text compression. Gao et al. (2005) describe an adaptive segmentation algorithm that allows for rapid retraining for new genres or segmentation standards and which does not assume a universal segmentation standard.

One of the significant challenges in comparing segmentation algorithms is the range in segmentation standards, and thus the lack of a common evaluation corpus, which would enable the direct comparison of algorithms. In response to this challenge, Chinese word segmentation has been the focus of several organized evaluations in recent years. The “First International Chinese Word Segmentation Bakeoff” in 2003 (Sproat and Emerson 2003), and several others since, have built on similar evaluations within China to encourage a direct comparison of segmentation methods. These evaluations have helped to develop consistent standards both for segmentation and for evaluation, and they have made significant contributions by cleaning up inconsistencies within existing corpora.

2.3.2.3 Japanese Segmentation

The Japanese writing system incorporates alphabetic, syllabic and logographic symbols. Modern Japanese texts, for example, frequently consist of many different writing systems: Kanji (Chinese Hanzi symbols), hiragana (a syllabary for grammatical markers and for words of Japanese origin), katakana (a syllabary for words of foreign origin), romanji (words written in the Roman alphabet), Arabic numerals, and various punctuation symbols. In some ways, the multiple character sets make tokenization easier, as transitions between character sets give valuable information about word boundaries. However, character set transitions are not enough, since a single word may contain characters from multiple character sets, such as inflected verbs, which can contain a Kanji base and hiragana inflectional ending. Company names also frequently contain a mix of Kanji and romanji. For these reasons, most previous approaches to Japanese segmentation, such as the popular JUMAN (Matsumoto and Nagao 1994) and Chasen programs (Matsumoto et al. 1997), rely on manually derived morphological analysis rules.

To some extent, Japanese can be segmented using the same statistical techniques developed for Chinese. For example, Nagata (1994) describes an algorithm for Japanese segmentation similar to that used

for Chinese segmentation by Sproat et al. (1996). More recently, Ando and Lee (2003) developed an

unsupervised statistical segmentation method based on n-gram counts in Kanji sequences that produces high performance on long Kanji sequences.

2.3.2.4 Unsegmented Alphabetic and Syllabic Languages

Common unsegmented alphabetic and syllabic languages are Thai, Balinese, Javanese, and Khmer. While such writing systems have fewer characters than Chinese and Japanese, they also have longer words; localized optimization is thus not as practical as in Chinese or Japanese segmentation. The richer morphology of such languages often allows initial segmentations based on lists of words, names, and affixes, usually using some variation of the maximum matching algorithm. Successful high-accuracy segmentation requires a thorough knowledge of the lexical and morphological features of the language. A nearly discussion of Thai segmentation can be found in Kawtrakul et al. (1996), describing a robust rule-based Thai segmenter and morphological analyzer. Meknavin et al. (1997) use lexical and collocational features automatically derived using machine learning to select an optimal segmentation from an n-best maximum matching set. Aroonmanakun (2002) uses a statistical Thai segmentation approach, which first seeks to segment the Thai text into syllables. Syllables are then merged into words based on a trained model of syllable collocation.

2.4 Sentence Segmentation

Sentences in most written languages are delimited by punctuation marks, yet the specific usage rules for punctuation are not always coherently defined. Even when a strict set of rules exists, the adherence to the rules can vary dramatically based on the origin of the text source and the type of text. Additionally, in different languages, sentences and subsentences are frequently delimited by different punctuation marks. Successful sentence segmentation for a given language thus requires an understanding of the various uses of punctuation characters in that language. In most languages, the problem of sentence segmentation reduces to disambiguating all instances of punctuation characters that may delimit sentences. The scope of this problem varies greatly by language, as does the number of different punctuation marks that need to be considered.

Written languages that do not use many punctuation marks present a very difficult challenge in recognizing sentence boundaries. Thai, for one, does not use a period (or any other punctuation mark) to mark sentence boundaries. A space is sometimes used at sentence breaks, but very often the space is indistinguishable from the carriage return, or there is no separation between sentences. Spaces are sometimes also used to separate phrases or clauses, where commas would be used in English, but this is also unreliable. In cases such as written Thai where punctuation gives no reliable information about sentence boundaries, locating sentence boundaries is best treated as a special class of locating word boundaries.

Even languages with relatively rich punctuation systems like English present surprising problems. Recognizing boundaries in such a written language involves determining the roles of all punctuation marks, which can denote sentence boundaries: periods, question marks, exclamation points, and sometimes semicolons, colons, dashes, and commas. In large document collections, each of these punctuation marks can serve several different purposes in addition to marking sentence boundaries. A period, for example, can denote a decimal point or a thousands marker, an abbreviation, the end of a sentence, or even an abbreviation at the end of a sentence. Ellipsis (a series of periods (...)) can occur both within sentences and at sentence boundaries. Exclamation points and question marks can occur at the end of a sentence, but also within quotation marks or parentheses (really!) or even (albeit infrequently) within a word, such as in the Internet company *Yahoo!* and the language name *!Xǎ*. However, conventions for the use of these two punctuation marks also vary by language; in Spanish, both can be unambiguously recognized as

sentence delimiters by the presence of 'j' or 'ζ' at the start of the sentence. In this section, we introduce the

challenges posed by the range of corpora available and the variety of techniques that have been successfully applied to this problem and discuss their advantages and disadvantages.

2.4.1 Sentence Boundary Punctuation

Just as the definition of what constitutes a sentence is rather arbitrary, the use of certain punctuation marks to separate sentences depends largely on an author's adherence to changeable and frequently ignored conventions. In most NLP applications, the only sentence boundary punctuation marks considered are the period, question mark, and exclamation point, and the definition of sentence is limited to the *text-sentence* (as defined by Nunberg 1990), which begins with a capital letter and ends in a full stop. However, grammatical sentences can be delimited by many other punctuation marks, and restricting sentence boundary punctuation to these three can cause an application to overlook many meaningful sentences or can unnecessarily complicate processing by allowing only longer, complex sentences. Consider Examples (6) and (7), two English sentences that convey exactly the same meaning; yet, by the traditional definitions, the first would be classified as two sentences, the second as just one. The semicolon in Example (7) could likewise be replaced by a comma or a dash, retain the same meaning, but still be considered a single sentence. Replacing the semicolon with a colon is also possible, though the resulting meaning would be slightly different.

(6) *Here is a sentence. Here is another.*

(7) *Here is a sentence; here is another.*

The distinction is particularly important for an application like part-of-speech tagging. Many taggers seek to optimize a tag sequence for a sentence, with the locations of sentence boundaries being provided to the tagger at the outset. The optimal sequence will usually be different depending on the definition of sentence boundary and how the tagger treats "sentence-internal" punctuation.

For an even more striking example of the problem of restricting sentence boundary punctuation, consider Example (8), from Lewis Carroll's *Alice in Wonderland*, in which *!?* are completely inadequate for segmenting the meaningful units of the passage:

(8) *There was nothing so VERY remarkable in that; nor did Alice think it so VERY much out of the way to hear the Rabbit say to itself, 'Oh dear! Oh dear! I shall be late!' (when she thought it over afterwards, it occurred to her that she ought to have wondered at this, but at the time it all seemed quite natural); but when the Rabbit actually TOOK A WATCH OUT OF ITS WAISTCOAT-POCKET, and looked at it, and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it, and fortunately was just in time to see it pop down a large rabbit-hole under the hedge.*

This example contains a single period at the end and three exclamation points within a quoted passage. However, if the semicolon and comma were allowed to end sentences, the example could be decomposed into as many as ten grammatical sentences. This decomposition could greatly assist in nearly all NLP tasks, since long sentences are more likely to produce (and compound) errors of analysis. For example, parsers consistently have difficulty with sentences longer than 15–25 words, and it is highly unlikely that any parser could ever successfully analyze this example in its entirety.

In addition to determining which punctuation marks delimit sentences, the sentence in parentheses as well as the quoted sentences *'Oh dear! Oh dear! I shall be late!'* suggest the possibility of a further decomposition of the sentence boundary problem into types of sentence boundaries, one of which would be "embedded sentence boundary." Treating embedded sentences and their punctuation differently could assist in the processing of the entire text-sentence. Of course, multiple levels of embedding would be

possible, as in Example (9), taken from *Watership Down* by Richard Adams. In this example, the main

sentence contains an embedded sentence (delimited by dashes), and this embedded sentence also contains an embedded quoted sentence.

(9) *The holes certainly were rough - "Just right for a lot of vagabonds like us," said Bigwig - but the exhausted and those who wander in strange country are not particular about their quarters.*

It should be clear from these examples that true sentence segmentation, including treatment of embedded sentences, can only be achieved through an approach, which integrates segmentation with parsing. Unfortunately, there has been little research in integrating the two; in fact, little research in computational linguistics has focused on the role of punctuation in written language.* With the availability of a wide range of corpora and the resulting need for robust approaches to NLP, the problem of sentence segmentation has recently received a lot of attention. Unfortunately, nearly all published research in this area has focused on the problem of sentence boundary detection in a small set of European languages, and all this work has focused exclusively on disambiguating the occurrences of period, exclamation point, and question mark. A great deal of recent work has focused on trainable approaches to sentence segmentation, which we discuss in Section 2.4.4. These new methods, which can be adapted to different languages and different text genres, should make a tighter coupling of sentence segmentation and parsing possible. While the remainder of this chapter focuses on published work that deals with the segmentation of a text into text-sentences, which represent the majority of sentences encountered in most text corpora, the above discussion of sentence punctuation indicates that the application of trainable techniques to broader problems may be possible. It is also important to note that this chapter focuses on disambiguation of punctuation in text and thus does not address the related problem of the *insertion* of punctuation and other structural events into automatic speech recognition transcripts of spoken language.

2.4.2 The Importance of Context

In any attempt to disambiguate the various uses of punctuation marks, whether in text-sentences or embedded sentences, some amount of the context in which the punctuation occurs is essential. In many cases, the essential context can be limited to the character immediately following the punctuation mark. When analyzing well-formed English documents, for example, it is tempting to believe that sentence boundary detection is simply a matter of finding a period followed by one or more spaces followed by a word beginning with a capital letter, perhaps also with quotation marks before or after the space. Indeed, in some corpora (e.g., literary texts) this single period-space-capital (or period-quote-space-capital) pattern accounts for almost all sentence boundaries. In *The Call of the Wild* by Jack London, for example, which has 1640 periods as sentence boundaries, this single rule correctly identifies 1608 boundaries (98%) (Bayer et al. 1998). However, the results are different in journalistic texts such as the *Wall Street Journal* (WSJ). In a small corpus of the WSJ from 1989 that has 16,466 periods as sentence boundaries, this simple rule would detect only 14,562 (88.4%) while producing 2900 **false positives**, placing a boundary where one does not exist.

Most of the errors resulting from this simple rule are cases where the period occurs immediately after an abbreviation. Expanding the context to consider whether the word preceding the period is a known abbreviation is thus a logical step. This improved abbreviation-period-space-capital rule can produce mixed results, since the use of abbreviations in a text depends on the particular text and text genre. The new rule improves performance on *The Call of the Wild* to 98.4% by eliminating five false positives (previously introduced by the phrase "St. Bernard" within a sentence). On the WSJ corpus, this new rule also eliminates all but 283 of the false positives introduced by the first rule. However, this rule also introduces 713 **false negatives**, erasing boundaries where they were previously correctly placed, yet still improving the overall score. Recognizing an abbreviation is therefore not sufficient to disambiguate the period, because we also must determine if the abbreviation occurs at the end of a sentence.

* A notable exception is Nunberg (1990).

The difficulty of disambiguating abbreviation-periods can vary depending on the corpus. Liberman and Church (Liberman and Church 1992) report that 47% of the periods in a *Wall Street Journal* corpus denote abbreviations, compared to only 10% in the Brown corpus (Francis and Kucera 1982), as reported by Riley (1989). In contrast, Müller et al. (1980) reports abbreviation-period statistics ranging from 54.7% to 92.8% within a corpus of English scientific abstracts. Such a range of figures suggests the need for a more informed treatment of the context that considers more than just the word preceding or following the punctuation mark. In difficult cases, such as an abbreviation which can occur at the end of a sentence, three or more words preceding and following must be considered. This is the case in the following examples of “garden path sentence boundaries,” the first consisting of a single sentence, the other of two sentences.

(10) *Two high-ranking positions were filled Friday by Penn St. University President Graham Spanier.*

(11) *Two high-ranking positions were filled Friday at Penn St. University President Graham Spanier announced the appointments.*

Many contextual factors have been shown to assist sentence segmentation in difficult cases. These contextual factors include

- **Case distinctions**—In languages and corpora where both uppercase and lowercase letters are consistently used, whether a word is capitalized provides information about sentence boundaries.
- **Part of speech**—Palmer and Hearst (1997) showed that the parts of speech of the words within three tokens of the punctuation mark can assist in sentence segmentation. Their results indicate that even an estimate of the *possible* parts of speech can produce good results.
- **Word length**—Riley (1989) used the length of the words before and after a period as one contextual feature.
- **Lexical endings**—Müller et al. (1980) used morphological analysis to recognize suffixes and thereby filter out words which were not likely to be abbreviations. The analysis made it possible to identify words that were not otherwise present in the extensive word lists used to identify abbreviations.
- **Prefixes and suffixes**—Reynar and Ratnaparkhi (1997) used both prefixes and suffixes of the words surrounding the punctuation mark as one contextual feature.
- **Abbreviation classes**—Riley (1989) and Reynar and Ratnaparkhi (1997) further divided abbreviations into categories such as titles (which are not likely to occur at a sentence boundary) and corporate designators (which are more likely to occur at a boundary).
- **Internal punctuation**—Kiss and Strunk (2006) used the presence of periods within a token as a feature.
- **Proper nouns**—Mikheev (2002) used the presence of a proper noun to the right of a period as a feature.

2.4.3 Traditional Rule-Based Approaches

The success of the few simple rules described in the previous section is a major reason sentence segmentation has been frequently overlooked or idealized away. In well-behaved corpora, simple rules relying on regular punctuation, spacing, and capitalization can be quickly written, and are usually quite successful. Traditionally, the method widely used for determining sentence boundaries is a regular grammar, usually with limited lookahead. More elaborate implementations include extensive word lists and exception lists to attempt to recognize abbreviations and proper nouns. Such systems are usually developed specifically for a text corpus in a single language and rely on special language-specific word lists; as a result they are not portable to other natural languages without repeating the effort of compiling extensive lists and rewriting rules. Although the regular grammar approach can be successful, it requires a large manual

effort to compile the individual rules used to recognize the sentence boundaries. Nevertheless, since

rule-based sentence segmentation algorithms can be very successful when an application does deal with well-behaved corpora, we provide a description of these techniques.

An example of a very successful regular-expression-based sentence segmentation algorithm is the text segmentation stage of the Alembic information extraction system (Aberdeen et al. 1995), which was created using the lexical scanner generator flex (Nicol 1993). The Alembic system uses flex in a preprocess pipeline to perform tokenization and sentence segmentation at the same time. Various modules in the pipeline attempt to classify all instances of punctuation marks by identifying periods in numbers, date and time expressions, and abbreviations. The preprocess utilizes a list of 75 abbreviations and a series of over 100 hand-crafted rules and was developed over the course of more than six staff months. The Alembic system alone achieved a very high accuracy rate (99.1%) on a large *Wall Street Journal* corpus. However, the performance was improved when integrated with the trainable system Satz, described in Palmer and Hearst (1997), and summarized later in this chapter. In this hybrid system, the rule-based Alembic system was used to disambiguate the relatively unambiguous cases, while Satz was used to disambiguate difficult cases such as the five abbreviations *Co.*, *Corp.*, *Ltd.*, *Inc.*, and *U.S.*, which frequently occur in English texts both within sentences and at sentence boundaries. The hybrid system achieved an accuracy of 99.5%, higher than either of the two component systems alone.

2.4.4 Robustness and Trainability

Throughout this chapter we have emphasized the need for robustness in NLP systems, and sentence segmentation is no exception. The traditional rule-based systems, which rely on features such as spacing and capitalization, will not be as successful when processing texts where these features are not present, such as in Example (1) above. Similarly, some important kinds of text consist solely of uppercase letters; closed captioning (CC) data is an example of such a corpus. In addition to being uppercase-only, CC data also has erratic spelling and punctuation, as can be seen from the following example of CC data from CNN:

(12) THIS IS A DESPERATE ATTEMPT BY THE REPUBLICANS TO SPIN THEIR STORY THAT NOTHING SEAR WHYOUS—SERIOUS HAS BEEN DONE AND TRY TO SAVE THE SPEAKER'S SPEAKERSHIP AND THIS HAS BEEN A SERIOUS PROBLEM FOR THE SPEAKER, HE DID NOT TELL THE TRUTH TO THE COMMITTEE, NUMBER ONE.

The limitations of manually crafted rule-based approaches suggest the need for trainable approaches to sentence segmentation, in order to allow for variations between languages, applications, and genres. Trainable methods provide a means for addressing the problem of embedded sentence boundaries discussed earlier, as well as the capability of processing a range of corpora and the problems they present, such as erratic spacing, spelling errors, single-case, and OCR errors.

For each punctuation mark to be disambiguated, a typical trainable sentence segmentation algorithm will automatically encode the context using some or all of the features described above. A set of training data, in which the sentence boundaries have been manually labeled, is then used to train a machine learning algorithm to recognize the salient features in the context. As we describe below, machine learning algorithms that have been used in trainable sentence segmentation systems have included neural networks, decision trees, and maximum entropy calculation.

2.4.5 Trainable Algorithms

One of the first published works describing a trainable sentence segmentation algorithm was Riley (1989). The method described used regression trees (Breiman et al. 1984) to classify periods according to contextual features describing the single word preceding and following the period. These contextual features included word length, punctuation after the period, abbreviation class, case of the word, and the

probability of the word occurring at beginning or end of a sentence. Riley's method was trained using 25

million words from the AP newswire, and he reported an accuracy of 99.8% when tested on the Brown corpus.

Palmer and Hearst (1997) developed a sentence segmentation system called Satz, which used a machine learning algorithm to disambiguate all occurrences of periods, exclamation points, and question marks. The system defined a contextual feature array for three words preceding and three words following the punctuation mark; the feature array encoded the context as the parts of speech, which can be attributed to each word in the context. Using the lexical feature arrays, both a neural network and a decision tree were trained to disambiguate the punctuation marks, and achieved a high accuracy rate (98%–99%) on a large corpus from the *Wall Street Journal*. They also demonstrated the algorithm, which was trainable in as little as one minute and required less than 1000 sentences of training data, to be rapidly ported to new languages. They adapted the system to French and German, in each case achieving a very high accuracy. Additionally, they demonstrated the trainable method to be extremely robust, as it was able to successfully disambiguate single-case texts and OCR data.

Reynar and Ratnaparkhi (1997) described a trainable approach to identify English sentence boundaries using a statistical maximum entropy model. The system used a system of contextual templates, which encoded one word of context preceding and following the punctuation mark, using such features as prefixes, suffixes, and abbreviation class. They also reported success in inducing an abbreviation list from the training data for use in the disambiguation. The algorithm, trained in less than 30 min on 40,000 manually annotated sentences, achieved a high accuracy rate (98%+) on the same test corpus used by Palmer and Hearst (1997), without requiring specific lexical information, word lists, or any domain-specific information. Though they only reported results on English, they indicated that the ease of trainability should allow the algorithm to be used with other Roman-alphabet languages, given adequate training data.

Mikheev (2002) developed a high-performing sentence segmentation algorithm that jointly identifies abbreviations, proper names, and sentence boundaries. The algorithm casts the sentence segmentation problem as one of disambiguating abbreviations to the left of a period and proper names to the right. While using unsupervised training methods, the algorithm encodes a great deal of manual information regarding abbreviation structure and length. The algorithm also relies heavily on consistent capitalization in order to identify proper names.

Kiss and Strunk (2006) developed a largely unsupervised approach to sentence boundary detection that focuses primarily on identifying abbreviations. The algorithm encodes manual heuristics for abbreviation detection into a statistical model that first identifies abbreviations and then disambiguates sentence boundaries. The approach is essentially language independent, and they report results for a large number of European languages.

Trainable sentence segmentation algorithms such as these are clearly necessary for enabling robust processing of a variety of texts and languages. Algorithms that offer rapid training while requiring small amounts of training data allow systems to be retargeted in hours or minutes to new text genres and languages. This adaptation can take into account the reality that good segmentation is task dependent. For example, in parallel corpus construction and processing, the segmentation needs to be consistent in both the source and target language corpus, even if that consistency comes at the expense of theoretical accuracy in either language.

2.5 Conclusion

The problem of text preprocessing was largely overlooked or idealized away in early NLP systems; tokenization and sentence segmentation were frequently dismissed as uninteresting. This was possible because most systems were designed to process small, monolingual texts that had already been manually selected, triaged, and preprocessed. When processing texts in a single language with predictable ortho-

graphic conventions, it was possible to create and maintain hand-built algorithms to perform tokenization

and sentence segmentation. However, the recent explosion in availability of large unrestricted corpora in many different languages, and the resultant demand for tools to process such corpora, has forced researchers to examine the many challenges posed by processing unrestricted texts. The result has been a move toward developing robust algorithms, which do not depend on the well-formedness of the texts being processed. Many of the hand-built techniques have been replaced by trainable corpus-based approaches, which use machine learning to improve their performance.

The move toward trainable robust segmentation systems has enabled research on a much broader range of corpora in many languages. Since errors at the text segmentation stage directly affect all later processing stages, it is essential to completely understand and address the issues involved in document triage, tokenization, and sentence segmentation and how they impact further processing. Many of these issues are language-dependent: the complexity of tokenization and sentence segmentation and the specific implementation decisions depend largely on the language being processed and the characteristics of its writing system. For a corpus in a particular language, the corpus characteristics and the application requirements also affect the design and implementation of tokenization and sentence segmentation algorithms. In most cases, since text segmentation is not the primary objective of NLP systems, it cannot be thought of as simply an independent “preprocessing” step, but rather must be tightly integrated with the design and implementation of all other stages of the system.

References

- Aberdeen, J., J. Burger, D. Day, L. Hirschman, P. Robinson, and M. Vilain (1995). MITRE: Description of the Alembic system used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Columbia, MD.
- Aho, A. V., R. Sethi, and J. D. Ullman (1986). *Compilers, Principles, Techniques, and Tools*. Reading, MA: Addison-Wesley Publishing Company.
- Ando, R. K. and L. Lee (2003). Mostly-unsupervised statistical segmentation of Japanese Kanji sequences. *Journal of Natural Language Engineering* 9, 127–149.
- Aroonmanakun, W. (2002). Collocation and Thai word segmentation. In *Proceedings of SNLP-COCOSDA2002*, Bangkok, Thailand.
- Baroni, M., F. Chantree, A. Kilgarriff, and S. Sharoff (2008). Cleaneval: A competition for cleaning web pages. In *Proceedings of the Sixth Language Resources and Evaluation Conference (LREC 2008)*, Marrakech, Morocco.
- Bayer, S., J. Aberdeen, J. Burger, L. Hirschman, D. Palmer, and M. Vilain (1998). Theoretical and computational linguistics: Toward a mutual understanding. In J. Lawler and H. A. Dry (Eds.), *Using Computers in Linguistics*. London, U.K.: Routledge.
- Breiman, L., J. H. Friedman, R. Olshen, and C. J. Stone (1984). *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group.
- Chang, P.-C., M. Galley, and C. D. Manning (2008). Optimizing Chinese word segmentation for machine translation performance. In *Proceedings of the Third Workshop on Statistical Machine Translation*, Columbus, OH, pp. 224–232.
- Comrie, B., S. Matthews, and M. Polinsky (1996). *The Atlas of Languages*. London, U.K.: Quarto Inc.
- Crystal, D. (1987). *The Cambridge Encyclopedia of Language*. Cambridge, U.K.: Cambridge University Press.

Daniels, P. T. and W. Bright (1996). *The World's Writing Systems*. New York: Oxford University Press. Francis, W. N. and H. Kucera (1982). *Frequency Analysis of English Usage*. New York: Houghton

Mifflin Co.

Fung, P. and D. Wu (1994). Statistical augmentation of a Chinese machine-readable dictionary. In *Proceedings of Second Workshop on Very Large Corpora (WVLC-94)*, Kyoto, Japan.

- Gao, J., M. Li, A. Wu, and C.-N. Huang (2005). Chinese word segmentation and named entity recognition: A pragmatic approach. *Computational Linguistics* 31(4), 531–574.
- Grefenstette, G. and P. Tapanainen (1994). What is a word, What is a sentence? Problems of Tokenization. In *The 3rd International Conference on Computational Lexicography (COMPLEX 1994)*, Budapest, Hungary.
- Hankamer, J. (1986). Finite state morphology and left to right phonology. In *Proceedings of the Fifth West Coast Conference on Formal Linguistics*, Stanford, CA.
- Hockenmaier, J. and C. Brew (1998). Error driven segmentation of Chinese. *Communications of COLIPS* 8(1), 69–84.
- Kawtrakul, A., C. Thumkanon, T. Jamjanya, P. Muangyunnan, K. Poolwan, and Y. Inagaki (1996). A gradual refinement model for a robust Thai morphological analyzer. In *Proceedings of COLING96*, Copenhagen, Denmark.
- Kiss, T. and J. Strunk (2006). Unsupervised multilingual sentence boundary detection. *Computational Linguistics* 32(4), 485–525.
- Liberman, M. Y. and K. W. Church (1992). Text analysis and word pronunciation in text-to-speech synthesis. In S. Furui and M. M. Sondhi (Eds.), *Advances in Speech Signal Processing*, pp. 791–831. New York: Marcel Dekker, Inc.
- Ma, Y. and A. Way (2009). Bilingually motivated domain-adapted word segmentation for statistical machine translation. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, Athens, Greece, pp. 549–557.
- Martin, J., H. Johnson, B. Farley, and A. MacLachlan (2003). Aligning and using an english-inuktitut parallel corpus. In *Proceedings of the HLT-NAACL 2003 Workshop on Building and Using Parallel Texts Data Driven: Machine Translation and Beyond*, Edmonton, Canada, pp. 115–118.
- Matsumoto, Y. and M. Nagao (1994). Improvements of Japanese morphological analyzer JUMAN. In *Proceedings of the International Workshop on Shareable Natural Language Resources*, Nara, Japan.
- Matsumoto, Y., A. Kitauchi, T. Yamashita, Y. Hirano, O. Imaichi, and T. Imamura (1997). Japanese morphological analysis system ChaSen manual. Technical Report NAIST-IS-TR97007, Nara Institute of Science and Technology, Nara, Japan (in Japanese).
- Meknavin, S., P. Charoenpornasawat, and B. Kijirikul (1997). Feature-based Thai word segmentation. In *Proceedings of the Natural Language Processing Pacific Rim Symposium 1997 (NLPRS97)*, Phuket, Thailand.
- Mikheev, A. (2002). Periods, capitalized words, etc. *Computational Linguistics* 28(3), 289–318.
- Müller, H., V. Amerl, and G. Natalis (1980). Worterkennungsverfahren als Grundlage einer Universal-methode zur automatischen Segmentierung von Texten in Sätze. Ein Verfahren zur maschinellen Satzgrenzenbestimmung im Englischen. *Sprache und Datenverarbeitung* 1.
- Nagata, M. (1994). A stochastic Japanese morphological analyzer using a Forward-DP backward A* n-best search algorithm. In *Proceedings of COLING94*, Kyoto, Japan.
- Nicol, G. T. (1993). *Flex—The Lexical Scanner Generator*. Cambridge, MA: The Free Software Foundation.
- Nunberg, G. (1990). *The Linguistics of Punctuation*. C.S.L.I. Lecture Notes, Number 18. Stanford, CA: Center for the Study of Language and Information.
- Palmer, D. D. (1997). A trainable rule-based algorithm for word segmentation. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL97)*, Madrid, Spain.
- Palmer, D. D. and M. A. Hearst (1997). Adaptive multilingual sentence boundary disambiguation. *Computational Linguistics* 23(2), 241–67.

- Park, Y. and R. J. Byrd (2001). Hybrid text mining for finding abbreviations and their definitions. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, Pittsburgh, PA.
- Reynar, J. C. and A. Ratnaparkhi (1997). A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the Fifth ACL Conference on Applied Natural Language Processing*, Washington, DC.

- Riley, M. D. (1989). Some applications of tree-based modelling to speech and language indexing. In *Proceedings of the DARPA Speech and Natural Language Workshop*, San Mateo, CA, pp. 339–352. Morgan Kaufmann.
- Sampson, G. R. (1995). *English for the Computer*. Oxford, U.K.: Oxford University Press.
- Sproat, R. and T. Emerson (2003). The first international Chinese word segmentation bakeoff. In *Proceedings of the Second SigHan Workshop on Chinese Language Processing*, Sapporo, Japan.
- Sproat, R. and C. Shih (2001). Corpus-based methods in Chinese morphology and phonology. Technical Report, Linguistic Society of America Summer Institute, Santa Barbara, CA.
- Sproat, R. W., C. Shih, W. Gale, and N. Chang (1996). A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics* 22(3), 377–404.
- Teahan, W.J., Y. Wen, R. McNab, and I. H. Witten (2000). A compression-based algorithm for Chinese word segmentation. *Computational Linguistics* 26(3), 375–393.
- Unicode Consortium (2006). *The Unicode Standard, Version 5.0*. Boston, MA: Addison-Wesley.
- Wu, A. (2003). Customizable segmentation of morphologically derived words in Chinese. *International Journal of Computational Linguistics and Chinese Language Processing* 8(1), 1–27.
- Wu, D. and P. Fung (1994). Improving Chinese tokenization with linguistic filters on statistical lexical acquisition. In *Proceedings of the Fourth ACL Conference on Applied Natural Language Processing*, Stuttgart, Germany.
- Wu, Z. and G. Tseng (1993). Chinese text segmentation for text retrieval: Achievements and problems. *Journal of the American Society for Information Science* 44(9), 532–542.
- Zhang, R., K. Yasuda, and E. Sumita (2008). Improved statistical machine translation by multiple Chinese word segmentation. In *Proceedings of the Third Workshop on Statistical Machine Translation*, Columbus, OH, pp. 216–223.

2. Literature Review

3

Lexical Analysis

3.1	Introduction	31
3.2	Finite State Morphology	33
	Closing Remarks on Finite State Morphology	
3.3	Finite State Morphology	37
	Disjunctive Affixes, Inflectional Classes, and Exceptionality · Further Remarks on Finite State Lexical Analysis	
3.4	“Difficult” Morphology and Lexical Analysis	42
	Isomorphism Problems · Contiguity Problems	
3.5	Paradigm-Based Lexical Analysis	47
	Paradigmatic Relations and Generalization · The Role of Defaults · Paradigm-Based Accounts of Difficult Morphology · Further Remarks on Paradigm-Based Approaches	
3.6	Concluding Remarks	54
	Acknowledgments	55
	References	55

Andrew Hippisley
University of Kentucky

3.1 Introduction

Words are the building blocks of natural language texts. As a proportion of a text’s words are morphologically complex, it makes sense for text-oriented applications to register a word’s structure. This chapter is about the techniques and mechanism for performing text analysis at the level of the word, *lexical analysis*. A word can be thought of in two ways, either as a string in running text, for example, the verb *delivers*; or as a more abstract object that is the cover term for a set of strings. So the verb DELIVER names the set {*delivers, deliver, delivering, delivered*}. A basic task of lexical analysis is to relate *morphological variants* to their *lemma* that lies in a lemma dictionary bundled up with its invariant semantic and syntactic information. Lemmatization is used in different ways depending on the task of the natural language processing

(NLP) system. In machine translation (MT), the lexical semantics of word strings can be accessed via the lemma dictionary. In transfer models, it can be used as part of the source language linguistic analysis to yield the morphosyntactic representation of strings that can occupy certain positions in syntactic trees, the result of syntactic analyses. This requires that lemmas are furnished not only with semantic but also with morphosyntactic information. So *delivers* is referenced by the item DELIVER + {3rd, Sg, Present}. In what follows we will see how the mapping between *deliver* and DELIVER, and the substrings *s* and {3rd, Sg, Present} can be elegantly handled using finite state transducers (FSTs).

We can think of the mapping of string to lemma as only one side of lexical analysis, the parsing side. The other side is mapping from the lemma to a string, morphological generation. Staying with our MT example, once we have morphosyntactically analyzed a string in the source language, we can then use the resulting information to generate the equivalent morphologically complex string in the target language. Translation at this level amounts to accessing the morphological rule of the target language that

introduces the particular set of features found from the source language parse. In information retrieval (IR), parsing and generation serve different purposes. For the automatic creation of a list of key terms, it makes sense to notionally collapse morphological variants under one lemma. This is achieved in practice during *stemming*, a text preprocessing operation where morphologically complex strings are identified, decomposed into invariant stem (= lemma's canonical form) and affixes, and the affixes are then deleted. The result is texts as search objects that consist of stems only so that they can be searched via a lemma list. Morphological generation also plays a role in IR, not at the preprocessing stage but as part of query matching. Given that a lemma has invariant semantics, finding an occurrence of one of its morphological variants satisfies the semantic demands of a search. In languages with rich morphology it is more economical to use rules to generate the search terms than list them. Moreover, since morphology is used to create new words through derivation, a text that uses a newly coined word would not be missed if the string was one of many outputs of a productive morphological rule operating over a given lemma. Spelling dictionaries also make use of morphological generation for the same reason, to account for both listed and 'potential' words. Yet another application of lexical analysis is text preprocessing for syntactic analysis where parsing a string into morphosyntactic categories and subcategories furnishes the string with POS tags for the input of a syntactic parse. Finally tokenization, the segmentation of strings into word forms, is an important preprocessing task required for languages without word boundaries such as Chinese since a morphological parse of the strings reveals morphological boundaries, including words boundaries.

It is important from the start to lay out three main issues that any lexical analysis has to confront in some way. First, as we have shown, lexical analysis may be used for generation or parsing. Ideally, the mechanism used for parsing should be available for generation, so that a system has the flexibility to go both ways. Most lexical analysis is performed using FSTs, as we will see. One of the reasons is that FSTs provide a trivial means from flipping from parsing (analysis) to generation. Any alternative to FST lexical analysis should at least demonstrate it has this same flexibility. Two further issues concern the linguistic objects of lexical analysis, morphologically complex words. The notion that they are structures consisting of an invariant stem encoding the meaning and syntactic category of a word, joined together with an affix that encodes grammatical properties such as number, person, tense, etc is actually quite idealistic. For some languages, this approach takes you a long way, for example, Kazakh, Finnish, and Turkish. But it needs refinement for the languages more often associated with large NLP applications such as English, French, German, and Russian.

One of the reasons that this is a somewhat idealized view of morphology is that morphosyntactic properties do not have to be associated with an affix. Compare, for example, the string *looked* which is analyzed as LOOK+{Past} with *sang*, also a simple past. How do you get from the string *sang* to the lemma SING+{Past, Simple}? There is no affix but instead an alternation in the canonical stem's vowel. A related problem is that the affix may be associated with more than one property set: *looked* may correspond to either LOOK+{Past, Simple} or LOOK+{Past, Participle}. How do you know which *looked* you have encountered? The second problem is that in the context of a particular affix, the stem is not guaranteed to be invariant, in other words equivalent to the canonical stem. Again not straying beyond English, the string associated with the lemma FLY+{Noun, Plural} is not **flys* but *flies*. At some level the parser needs to know that *flie* is part of the FLY lemma, not some as yet unrecorded FLIE lemma; moreover this variant form of the stem is constrained to a particular context, combination with the suffix *-s*. A further complication is changes to the canonical affix. If we propose that *-s* is the (orthographic) plural affix in English we have to account for the occasions when it appears in a text as *-es*, for example, in *foxes*.

In what follows we will see how lexical analysis models factor in the way a language assigns structure to words. Morphologists recognize three main approaches to word structure, first discussed in detail in Hockett (1958) but also in many recent textbooks, for example, Booij (2007: 116–117). All three approaches find their way into the assumptions that underlie a given model. An item and arrangement approach (I&A) views analysis as computing the information conveyed by a word's stem morpheme with that of its affix morpheme. Finite state morphology (FSM) incorporates this view using FSTs. This works

well for the 'ideal' situation outlined above: *looked* is a stem plus a suffix, and information that the word

conveys is simply a matter of computing the information conveyed by both morphemes. Item and process approaches (I&P) account for the kind of stem and affix variation that can happen inside a complex word, for example, *sing* becomes *sang* when it is past tense, and a vowel is added to the suffix *-s* when attached to *fox*. The emphasis is on possible phonological processes that are associated with affixation (or other morphological operations), what is known as *morphonology*. Finally, in word and paradigm approaches (W&P), a lemma is associated with a table, or paradigm, that associates a morphological variant of the lemma with a morphosyntactic property set. So *looked* occupies the cell in the paradigm that contains the pairing of LOOK with {Past, Simple}. And by the same token *sang* occupies the equivalent cell in the SING paradigm. Meaning is derived from the definition of the cell, not the meaning of stem plus meaning of suffix, hence no special status is given to affixes.

FSTs have been used to handle *morphonology*, expressed as spelling variation in a text, and *morphotactics*, how stems and affixes combine, and how the meaning behind the combination can be computed. We begin with FSTs for morphonology, the historic starting point for FSM. This leaves us clear to look at lexical analysis as morphology proper. We divide this into two main parts, the model that assumes the I&A approach using FSTs (Section 3.3) and the alternative W&P model (Section 3.5). Section 3.4 is a brief overview of the types of ‘difficult’ morphology that the paradigm-based approaches are designed to handle but which FSM using the I&A approach can negotiate with some success too.

3.2 Finite State Morphonology

Phonology plays an important role in morphological analysis, as affixation is the addition of phonological segments to a stem. This is phonology as *exponent* of some property set. But there is another ‘exponent-less’ way in which phonology is involved, a kind of phonology of morpheme boundaries. This area of linguistics is known as *morphophonology* or *morphonology*: “the area of linguistics that deals with the relations and interaction of morphology with phonology” (Aronoff and Fudeman 2005: 240). Morpheme boundary phonology may or may not be reflected in the orthography. For example, in Russian word final voiced obstruents become voiceless – but they are spelled as if they stay as they are, unvoiced. A good example of morphonology in English is plural affixation. The plural affix can be pronounced in three different ways, depending on the stem it attaches to: as /z/ in *flags*, as /z/ in *glasses* and as /s/ in *cats*. But only the /z/ alternation is consequential because it shows up as a variant of orthographic *-s*. Note that text to speech processing has to pay closer attention to morphonology since it has to handle the two different pronunciations of orthographic *-s*, and for the Russian situation it has to handle the word final devoicing rule. For lexical analysis to cope with morphonological alternations, the system has to provide a means of mapping the ‘basic’ form with its orthographic variant. As the variation is (largely) determined by context, the mapping can be rule governed. For example, the suffix *-s* you get in a plural word shows up as *-es* (the mapping) when the stem it attaches to ends in a *-s-* (specification of the environment). As we saw in the previous section, stems can also have variants. For *flie* we need a way of mapping it to basic *fly*, and a statement that we do this every time we see this string with a *-s* suffix. Note that this is an example of orthographic variation with no phonological correlate (*flie* and *fly* are pronounced the same).

The favored model for handling morphonology in the orthography, or morphology-based orthographic spelling variation, is a specific type of finite state machine known as a finite state transducer (FST). It is assumed that the reader is familiar with finite state automata. Imagine a finite state transition network (FSTN) which takes two tapes as input, and transitions are licensed not by arcs notated with a single symbol but a pair of symbols. The regular language that the machine represents is the *relation* between the language that draws from one set of symbols and the language that draws from the set of symbols it is paired with. An FST that defines the relation between underlying *glass*[^]s (where [^] marks a morpheme boundary) and surface *glasses* is given in Figure 3.1.

Transition from one state to another is licensed by a specific correspondence between symbols belonging

to two tapes. Underlying, more abstract representations are conventionally the upper tape. The colon

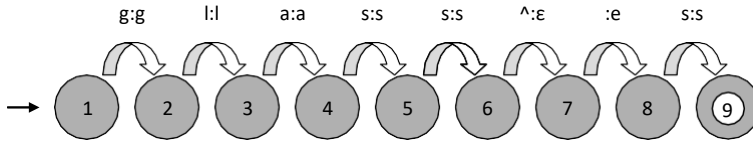


FIGURE 3.1 A spelling rule FST for *glasses*.

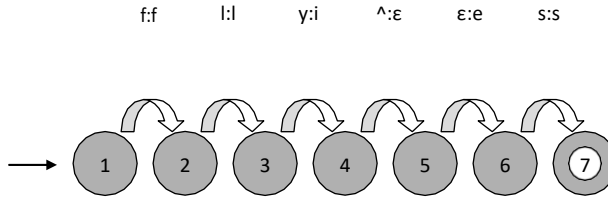


FIGURE 3.2 A spelling rule FST for *flies*.

between symbols labeling the arcs declares the correspondence. The analysis of the surface string into its canonical morphemes is simply reading the lower language symbol and printing its upper language correspondent. And generation is the inverse. Morpheme boundaries do not have surface representations; they are deleted in generation by allowing the alphabet of the lower language to include the empty string symbol ϵ . This correspondence of $\wedge$ to ϵ labels the transition from State 6 to State 7. The string *glasses* is an example of *insertion*-based orthographic variation where a character has to be inserted between stem and suffix. Since ϵ can also belong to the upper language its correspondence with lower language e provides for the insertion (State 7 to State 8). In a similar vein, an FST can encode the relation between underlying *fly*[^]s and surface *flies* (Figure 3.2). This is a combination of *substitution* based variation (the symbol y is substituted by i) and insertion based variation, if we treat the presence of e as the same as in the *glasses* example. Variation takes place both in the stem and in the suffix.

A practical demonstration of an FST treatment of English orthographic variation, i.e., spelling rules helps to show these points. To do this we will use the lexical knowledge representation language DATR (Evans and Gazdar 1996). DATR notation for FSTs is a good expository choice since its syntax is particularly transparent, and has been shown to define FSTs in an economical way (Evans and Gazdar 1996: 191–193).^{*} And as we will be using DATR anyway when we discuss an alternative to finite state-based lexical analysis, it makes sense to keep with the same notation throughout. But the reader should note that there are alternative FST notations for lexical analysis, for example, in Koskenniemi (1983), Beesley and Karttunen (2003), and Sproat (1997). DATR expresses the value for some attribute, or a set of attributes, as an association of the value with a path at some node. A DATR definition is given in (3.1).

$$(3.1) \quad \text{State\_n:} \\ \quad \langle g \rangle == g.$$

Basically, (3.1) says that at a particular node, *State_n*, the attribute $\langle g \rangle$ has the value g . Nodes are in (initial) upper case, attribute sets are paths of one or more atoms delimited by angle brackets. We could think of (3.1) as a trivial single state FST that takes an input string g , represented by an attribute path, and generates the output string g , represented by the value. DATR values do not need to be explicitly stated, they can be *inherited* via another path. And values do not need to be simple; they can be a combination of atom(s) plus inheriting path(s). Imagine you are building a transducer to transliterate words in the Cyrillic alphabet into their Roman alphabet equivalents. For example, you want the FST to capture the proper name *Caxa* transliterated as *Sasha*. So we would have $\langle C \rangle == S$, $\langle a \rangle == a$. For x we need two glyphs and to get them we could associate $\langle x \rangle$ with the complex value $s \langle h \rangle$, and somewhere else provide the equation $\langle h \rangle == h$. So $\langle x \rangle == s \langle h \rangle$ implies $\langle x \rangle == s h$. We will

* Gibbon (1987) is an FST account of tone morphonology in Tem and Baule, African languages spoken in Togo and the Ivory Coast. In a later demonstration, Gibbon showed how DATR could be used to considerably reduce the number of states needed to describe the same problem (Gibbon 1989).

see the importance of including a path as part of the value to get (3.1) to look more like a more serious transducer that maps $glass^s$ to $glasses$. This is given in (3.2).

```
(3.2)  Glasses_FST:
        <g> == g <>
        <l> == l <>
        <a> == a <>
        <s> == s <>
        <s> == s <>
        <^> == e <>
        <s> == s <>
        <> == .
```

The input string is the path $\langle g \ l \ a \ s \ s \ ^ \ s \rangle$. The path $\langle g \rangle$ that we see in the second line of (3.2) is in fact the *leading subpath* of this path. The leading subpath expresses the first symbol of the input string. It is associated with the atom g , a symbol of the output string. So far, we have modeled a transition from the initial state to another state, and transduced g to g . Further transitions are by means of the $\langle \rangle$ and this needs careful explanation. In DATR, any extensions of a subpath on the left of an equation are automatically transferred to a path on the right of the equation. So the extensions of $\langle g \rangle$ are transferred into the path $\langle \rangle$ as $\langle l \ a \ s \ s \ ^ \ s \rangle$. This path then needs to be evaluated by linking it to a path on the left hand side. The path $\langle l \rangle$ in the third line is suitable because it is the leading subpath of this new path. As we can see the value associated with $\langle l \rangle$ is the atom l , so another symbol has been consumed on the input string and a corresponding symbol printed onto the output string. The extensions of $\langle l \ a \ s \ s \ ^ \ s \rangle$ fill the path $\langle \rangle$ on the right side. To take stock: at this point the evaluation of $\langle g \ l \ a \ s \ s \ ^ \ s \rangle$ is $g \ l$ together with the extended path $\langle a \ s \ s \ ^ \ s \rangle$. As we continue down the equation list the leading subpaths are always the next attribute atom in the input string path, and this path is given the equivalent value atom. But something more interesting happens when we get to the point where the leading subpath is $\langle ^ \rangle$. Here a nonequivalent value atom is given, the atom e . This of course expresses the e insertion that is the essence of the spelling rule. The deletion of the $\wedge$ is represented very straightforwardly as saying nothing about it, i.e., no transduction. Finally, the equation at the bottom of the list functions to associate any subpath not already specified, expressed as $\langle \rangle$, with a null value. Suppose we represent input strings with an end of word boundary $\#$, so we have the lexical entry $\langle g \ l \ a \ s \ s \ ^ \ e \ s \ \# \rangle$. Through the course of the evaluation $\langle \# \rangle$ will ultimately be treated as a leading subpath. As this path is not explicitly stated anywhere else at the node, it is implied by $\langle \rangle$. So $\langle \rangle ==$ is interpreted as the automatic deletion of any substring for which there is no explicit mapping statement. This equation also expresses the morphologically simple input string $\langle g \ l \ a \ s \ s \ \# \rangle$ mapping to $g \ l \ a \ s \ s$. The theorem, expressing input and output string correspondences licensed by the FST in (3.2) is given in (3.3).

```
(3.3)  <g l a s s #> = g l a s s
        <g l a s s ^ s #> = g l a s s e s
```

The FST in (3.2) is very useful for a single word in the English language but says nothing about other words, such *class:classes*, *mass:masses*, or *fox:foxes*. Nor does it provide for 'regular' plurals such as *cat:cats*. FSTs are set up to manage the regular situation as well as problems that are general to entire classes. To do this, symbols can be replaced by *symbol classes*. (3.4) replaces (3.3) by using a symbol class represented by the expression $\$abc$, an abbreviatory variable ranging over the 26 lower case alphabetic characters used in English orthography (see Evans and Gazdar 1996: 192–193 for the DATR FST on which this is based).

```
(3.4)  Glasses&Classes:
        <$abc> == $abc <>
```

$$\langle \hat{\cdot} \rangle == e \langle \cdot \rangle$$

$$\langle \cdot \rangle == .$$

For an input string consisting of a stem composed of alphabetic symbols, the first equation takes this string represented as a path and associates whatever character denotes its leading subpath with the equivalent character as an atomic value. Equivalence is due to the fact that $\$abc$ is a bound variable. If the string is the path $\langle g \ 1 \ a \ s \ s \ ^ \ s \ \# \rangle$ then the leading subpath $\langle g \rangle$ is associated with the atomic value g ; by the same token for the string $\langle c \ a \ t \ ^ \ s \rangle$ the subpath $\langle c \rangle$ would be associated with c . The extension of this path fills $\langle \rangle$ on the right hand side as in (3.2), and just in case the new leading subpath belongs to $\$abc$, it will be evaluated as $\langle \$abc \rangle == \&abc \ \langle \rangle$. This represents a self-loop, a transition whose source and destination state is the same. In case we hit a morpheme boundary, i.e., we get to the point where the leading subpath is $\langle ^ \rangle$, then as in (3.2) the value given is e . Whatever appears after the morpheme boundary is the new leading subpath. And since it is $\langle s \rangle$, the plural affix, it belongs to $\$abc$ so through $\langle \$abc \rangle == \$abc \ \langle \rangle$ will map to s . As before, the $\#$ will be deleted through $\langle \rangle ==$ since this symbol does not belong to $\$abc$.

Whereas (3.2) undergeneralizes, we now have an FST in (3.4) which overgeneralizes. If the input is $\langle c \ a \ t \ ^ \ s \rangle$ the output will be incorrect $c \ a \ t \ e \ s$. A morphonological rule or spelling rule (the orthographic counterpart) has to say not only (a) what changes from one level representation to another, and (b) where in the string the change takes place but also (c) under what circumstances. The *context* of e insertion is an s followed by the symbol sequence $^ \ s$. But if we want to widen the context so that *foxes* is included in the rule, then the rule needs to specify e insertion when not just s but also x is followed by $^ \ s$. We can think of this as a class of contexts, a subset of the stem symbol class above. Figure 3.3 is a graphical representation of a transition labeled by the symbol class of all stem characters, and another transition labeled by the class of just those symbols providing the left context for the spelling rule.

Our (final) FST for the *-es* spelling rule in English is given in (3.5) with its theorem in (3.6). The context of e insertion is expressed by the variable $\$sx$ (left context) followed by the morpheme boundary symbol (right context). As 'regular' input strings such as $\langle c \ a \ t \ ^ \ s \rangle$, $\langle d \ o \ g \ ^ \ s \rangle$,

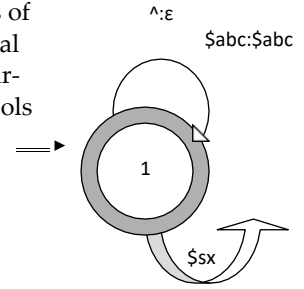


FIGURE 3.3 An FST with symbol classes.

$\langle c \ h \ a \ i \ r \ ^ \ s \rangle$ do not have pre-morpheme boundary s or x they avoid the path leading to e insertion.

(3.5) $Es\_Spelling\_Rule:$
 $\langle \$abc \rangle == \$abc \ \langle \rangle$
 $\langle \$sx \ ^ \rangle == \$sx \ e \ \langle \rangle$
 $\langle ^ \rangle == \langle \rangle$
 $\langle \rangle == .$

(3.6) $\langle g \ 1 \ a \ s \ s \ \# \rangle = g \ 1 \ a \ s \ s.$
 $\langle g \ 1 \ a \ s \ s \ ^ \ s \ \# \rangle = g \ 1 \ a \ s \ s \ e \ s.$
 $\langle f \ o \ x \ \# \rangle = f \ o \ x.$
 $\langle f \ o \ x \ ^ \ s \rangle = f \ o \ x \ e \ s.$
 $\langle c \ a \ t \ \# \rangle = c \ a \ t.$
 $\langle c \ a \ t \ ^ \ s \ \# \rangle = c \ a \ t \ s.$

A final comment on the spelling rule FST is in order. In (3.5) how do we ensure that the subpath $\langle x \rangle$ for $\langle f \ o \ x \ ^ \ s \rangle$ will not be evaluated by the first equation since x belongs to $\$abc$ as well as $\$sx$? In other words, how do we 'look ahead' to see if the next symbol on the input string is a $^$? In DATR, look ahead is captured by the 'longest path wins' principle so that any extension of a subpath takes precedence

over the subpath. As $\langle x^{\wedge} \rangle$ is an extension of $\langle x \rangle$, the $\langle x^{\wedge} \rangle$ path ‘wins’ and gets evaluated, i.e., it overrides the shorter path and its value. We look more closely at this principle when we use DATR to represent default inheritance hierarchies in Section 5.

3.2.1 Closing Remarks on Finite State Morphonology

Morphonological alternations at first glance seem marginal to word structure, or morphology ‘proper.’ In the previous discussion, we have barely mentioned a morphosyntactic feature. But their importance in lexical analysis should not be overlooked. On the one hand, text processors have to somehow handle orthographic variation. And on the other, it was earlier attempts at computationally modeling theoretical accounts of phonological and morphonological variations that suggested FSTs were the most efficient means of doing this. Kaplan and Kay in the 1980s, belatedly published as Kaplan and Kay (1994), demonstrated that the (morph)phonological rules for English proposed by Chomsky and Halle (1968) could be modeled in FSTs. Their work was taken up by Koskenniemi (1983) who used FSTs for the morphonology of Finnish, which went beyond proof of concept and was used in a large-scale text-processing application. Indeed Koskenniemi’s Two-Level Morphology model is the real starting point for finite state-based analysis. Its motivation was to map the underlying lexical (= lemma) representation to the surface representation without the need to consult an intermediary level. Indeed, intermediary levels can be handled by cascading FSTs so that the output of FST1 is the input of FST2, and the output of FST2 is in the input of FST3. But then the ordering becomes crucial for getting the facts right. Koskenniemi had the FSTs operate in parallel. An FST requires a particular context that could be an underlying or surface symbol (class) and specifies a particular mapping between underlying and surface strings. It thus acts as a constraint on the mapping of underlying and surface representations, and the specific environment of this mapping. All FSTs simultaneously scan both underlying and surface strings. A mapping is accepted by all the FSTs that do not specify a constraint. For it to work the underlying and surface strings have to be equal length, so the mapping is one to one. One rule maps underlying *y* to surface *i* provided that a surface *e* comes next; so the context is the surface string. The other is sensitive to the underlying string where it ensures a surface *e* appears whenever *y* precedes the morpheme boundary, shown in (3.6).

(3.6)

f	l	y	0	^	s	
f	l	i	e	0	s	

Koskenniemi’s model launched FST-based morphology because, as Karttunen (2007: 457) observes, it was “the first practical model in the history of computational linguistics for analysis of morphologically complex languages.” Despite its title, the framework was essentially for morphonology rather than morphology proper, as noted in an early review (Gazdar 1985: 599). Nonetheless, FST morphonology paved the way for FST morphology proper which we now discuss.

3.3 Finite State Morphology

In the previous section we showed how lexical analysis has to account for surface variation of a canonical string. But the canonical string with morpheme boundaries is itself the lower string of its associated lemma. For example, *fox*’s has the higher-level representation as the (annotated) lemma *fox+noun^plural*. FSTs are used to translate between these two levels to model what we could think of as morphology ‘proper.’ To briefly highlight the issues in FSM let us consider an example from Turkish with a morphosyntactic translation, or interlinear gloss, as well as a standard translation.

(3.7)

gör-mü-yor-du-k
see-NEG-PROGR-PAST-1PL
‘We weren’t seeing’ (Mel’čuk 2006: 299)

In Turkish, the morphological components of a word are neatly divisible into stem and contiguous affixes

where each affix is an exponent of a particular morphosyntactic property. Lexical analysis treats the interlinear gloss (second line) as the lemma and maps it onto a morphologically decomposed string. The

language of the upper, or lexical, language contains symbols for morphosyntactic features. The ordering of the morphemes is important: Negation precedes Aspect which precedes Tense which in turn precedes Subject Agreement information. For a correct mapping, the FST must encode morpheme ordering, or *morphotactics*. This is classic I&A morphological analysis. As in the previous section, we can demonstrate with an FST for English notated in DATR (3.8). English does not match Turkish for richness in inflectional morphology but does better in derivational morphology. The lexical entries for the derivational family *industry*, *industrial*, *industrialize*, *industrialization* are given in (3.8b).

```
(3.8)  DERIVATION:
        <$abc> == $abc <>
        <+noun> == Noun_Stem:<>.
Noun_Stem:
        <> == \#
        <^ +adj> == ^ a l Adj_Stem:<>.
Adj_Stem:
        <> == \#
        <^ +vb> == ^ i z e Verb_Stem:<>.
Verb_Stem:
        <> == \#
        <^ +noun> == ^ a t i o n <>.
```

```
(3.8b) <i n d u s t r y +noun>
        <i n d u s t r y +noun ^ +adj>
        <i n d u s t r y +noun ^ +adj ^ +vb>
        <i n d u s t r y +noun ^ +adj ^ +vb ^ +noun>.
```

The FST maps the lemma lexical entries in (3.8b) to their corresponding (intermediary) forms, the noun *industry*#, the adjective *industry*^{al}#, the verb *industry*^{al}*ize*#, and the noun *industry*^{al}*ize**ation*#. As in the morphonological demonstration in the previous section, the trivial alphabetical mapping is performed through a variable expressing a symbol class and path extensions for arc transitioning. The first difference is a path with a morphosyntactic feature as its attribute, <+noun> showing that in this FST we have lemmas and features as input. We see that this feature licenses the transition to another set of states gathered round the node *Noun_Stem*. In FSM, lemmas are classified according to features such as POS to enable appropriate affix selection, and hence capture the morphotactics of the language. Three nodes representing three stem classes are associated with the three affixes *-al*, *-ize*, *-ation*. For ^{a l} to be a possible affix value the evaluation must be at the *Noun_Stem* node. Once the affix is assigned, further evaluation must be continued at a specified node, here *Adj_Stem*. This is because the continuation to *-al* affixation is severely restricted in English. We can think of the specified ‘continuation’ node as representing a *continuation class*, a list of just those affixes that can come after *-al*. In this way, a lemma is guided through the network, outputting an affix and being shepherd to the subnetwork where the next affix will be available. So (3.8) accounts for *industry*^{al}*ize**ation*# but fails for **industry*^{ation}^{al}*ize*# or **industry-ize-al-ation*#. It also accounts for *industry*# and *industry*^{al}*ize*# by means of the equation <> == \ # at each continuation class node. Note that # is a reserved symbol in DATR, hence the need for escape \.

Let us quickly step through the FST to see how it does the mapping <i n d u s t r y +noun ^ +adj ^ +vb> = i n d u s t r y ^ a l ^ i z e #. The first path at DERIVATION maps the entire stem of the lemma to its surface form, in the manner described for the spelling rule FST. After this, the leading subpath is <+noun>; the path extensions are passed over to the node *Noun_Stem*. The first line at *Noun_Stem* covers the morphologically simple <i n d u s t r y +noun>. For this string, there is no further path to extend, i.e., no morphological boundaries, and transduction

amounts to appending to the output string the word boundary symbol. Similar provision is made at all

nodes just in case the derivation stops there. If, however, the lemma is annotated as being morphologically complex, and specifically as representing adjectival derivation, $\langle^+ \text{adj}\rangle$, the output is a morpheme boundary plus *-al* affix (second line at the `Adj_Stem` node). At this point the path can be extended as $\langle^+ \text{vb}\rangle$ in the case of derivative *industrialize* or *industrialization*, or not in the case of *industrial*. With no extensions, evaluation will be through $\langle\rangle == \backslash\#$ yielding `i n d u s t r y ^ a l \#`. Otherwise an extension with leading subpath $\langle^+ \text{vb}\rangle$ outputs suffix `^ i z e` and is then passed onto the node `Verb_Stem` for further evaluation. As there is no new subpath the end of word boundary is appended to the output string value. But if the input path happened to extend this path any further evaluation would have to be at `Verb_Stem`, e.g., adding the affix *-ation*.

3.3.1 Disjunctive Affixes, Inflectional Classes, and Exceptionality

Affix continuation classes are important for getting the morphotactics right but they also allow for more than one affix to be associated with the same morphosyntactic feature set. This is very common in inflectionally rich languages such as Russian, French, Spanish, and German. To illustrate, consider the paradigm of the Russian word *karta* ‘map.’ I am giving the forms in their transliterated versions for expository reasons, so it should be understood that *karta* is the transliteration of $\kappa \text{ap} \tau \text{a}$. Note the suffix used for the genitive plural $-\emptyset$. This denotes a ‘zero affix,’ i.e., the word is just the stem *kart* (or $\kappa \text{ap} \tau$) in a genitive plural context.

(3.9)	Karta		
		Singular	Plural
	Nominative	<i>kart-a</i>	<i>kart-y</i>
	Accusative	<i>kart-u</i>	<i>kart-y</i>
	Genitive	<i>kart-y</i>	<i>kart-$\emptyset$</i>
	Dative	<i>kart-e</i>	<i>kart-am</i>
	Instrumental	<i>kart-oj</i>	<i>kart-ami</i>
	Locative	<i>kart-e</i>	<i>kart-ax</i>

The FST in (3.10) maps lexical entries such as $\langle \text{k a r t} + \text{noun} \wedge \text{sg nom} \rangle$ to its corresponding surface form `k a r t ^ a \#`.

(3.10)	RUSSIAN:
	$\langle \$abc \rangle == \$abc \langle \rangle$
	$\langle +\text{noun} \rangle == \text{Noun\_Stem} : \langle \rangle$
	Noun_Stem:
	$\langle \rangle == \backslash\#$
	$\langle^+ \text{sg nom} \rangle == ^ a \langle \rangle$
	$\langle^+ \text{sg acc} \rangle == ^ u \langle \rangle$
	$\langle^+ \text{sg gen} \rangle == ^ y \langle \rangle$
	$\langle^+ \text{sg dat} \rangle == ^ e \langle \rangle$
	$\langle^+ \text{sg inst} \rangle == ^ o j \langle \rangle$
	$\langle^+ \text{sg loc} \rangle == ^ e \langle \rangle$
	$\langle^+ \text{pl nom} \rangle == ^ y \langle \rangle$
	$\langle^+ \text{pl acc} \rangle == ^ y \langle \rangle$
	$\langle^+ \text{pl gen} \rangle == ^ 0 \langle \rangle$
	$\langle^+ \text{pl dat} \rangle == ^ am \langle \rangle$
	$\langle^+ \text{pl inst} \rangle == ^ a m i \langle \rangle$
	$\langle^+ \text{pl loc} \rangle == ^ a x \langle \rangle$

This FST accounts for *any* Russian noun. But this makes it too powerful as not all nouns share the

inflectional pattern of *karta*. For example, *zakon* 'law' has a different way of forming the genitive

TABLE 3.1 Russian Inflectional Classes

	I Zakon	II Karta	III Rukopis'	IV Boloto
Singular				
Nom	<i>zakon-ø</i>	<i>kart-a</i>	<i>rukopis'-ø</i>	<i>bolot-o</i>
Acc	<i>zakon-ø</i>	<i>kart-u</i>	<i>rukopis'-ø</i>	<i>bolot-o</i>
Gen	<i>zakon-a</i>	<i>kart-y</i>	<i>rukopis-i</i>	<i>bolot-a</i>
Dat	<i>zakon-u</i>	<i>kart-e</i>	<i>rukopis-i</i>	<i>bolot-u</i>
Inst	<i>zakon-om</i>	<i>kart-oj</i>	<i>rukopis-ju</i>	<i>bolot-om</i>
Loc	<i>zakon-e</i>	<i>kart-e</i>	<i>rukopis-i</i>	<i>bolot-e</i>
Plural				
Nom	<i>zakon-y</i>	<i>kart-y</i>	<i>rukopis-i</i>	<i>bolot-a</i>
Acc	<i>zakon-y</i>	<i>kart-y</i>	<i>rukopis-i</i>	<i>bolot-a</i>
Gen	<i>zakon-ov</i>	<i>kart-ø</i>	<i>rukopis-ej</i>	<i>bolot-ø</i>
Dat	<i>zakon-am</i>	<i>kart-am</i>	<i>rukopis-jam</i>	<i>bolot-am</i>
Inst	<i>zakon-ami</i>	<i>kart-ami</i>	<i>rukopis-jami</i>	<i>bolot-ami</i>
Loc	<i>zakon-ax</i>	<i>kart-ax</i>	<i>rukopis-jax</i>	<i>bolot-ax</i>

singular: it affixes *-a* to the stem and not *-y* (*zakon-a*). And *bolot-o* 'swamp' differs in its nominative singular. Finally *rukopis'* 'manuscript' has a distinct dative singular *rukopisi*. Because of these and other distinctions, Russian can be thought of as having four major inflectional patterns, or *inflectional classes*, that are shown in Table 3.1.

To handle situations where there is a choice of affix corresponding to a given morphosyntactic property set, an FST encodes subclasses of stems belonging to the same POS class. (3.11) is a modified version of the FST in (3.10) that incorporates inflectional classes as sets of disjunctive affixes. For reasons of space, only two classes are represented. Sample lexical entries are given in (3.12).

```
(3.11)  RUSSIAN_2:
        <$abc> == $abc <>
        <+noun> == Noun_Stem:<>.
Noun_Stem:
    < 1 > == Stem_1:<>

    < 2 > == Stem_2:<>

    < 3 > == Stem_3:<>

    < 4 > == Stem_4:<>.

Stem_1:
    <> == \#
    <^ sg nom> == ^ 0 <>
    <^ sg acc> == ^ 0 <>
    <^ sg gen> == ^ a <>
    <^ sg dat> == ^ u <>
    <^ sg inst> == ^ o m <>
    <^ sg loc> == ^ e <>
    <^ pl nom> == ^ y <>
    <^ pl acc> == ^ y <>
```

```
<^ pl gen> == ^ o v <>  
<^ pl dat> == ^ a m <>  
<^ pl inst> == ^ a m i <>  
<^ pl loc> == ^ a x <>.
```

```

Stem_2:
<> == \#
<^ sg nom> == ^ a <>
<^ sg acc> == ^ u <>
<^ sg gen> == ^ y <>
<^ sg dat> == ^ e <>
<^ sg inst> == ^ o j <>
<^ sg loc> == ^ e <>
<^ pl nom> == ^ y <>
<^ pl acc> == ^ y <>
<^ pl gen> == ^ 0 <>
<^ pl dat> == ^ am <>
<^ pl inst> == ^ a m i <>
<^ pl loc> == ^ a x <>.

```

(3.12) <k a r t +noun 2 ^ sg nom>
 <k a r t +noun 2 ^ sg acc>
 <z a k o n +noun 1 ^ sg nom>
 <z a k o n +noun 1 ^ sg acc>

What is different about the partial lexicon in (3.12) is that stems are annotated for stem class (1, 2, 3, 4) as well as POS. The node *Noun_Stem* assigns stems to appropriate stem class nodes for affixation. Each of the four stem class nodes maps a given morphosyntactic feature sequence to an affix. In this way, separate affixes that map to a single feature set do not compete as they are distributed across the stem class nodes. Even English has something like inflectional classes. There are several ways of forming a past participle: suffix *-ed* as in 'have looked,' suffix *-en* as in 'have given,' and no affix (*-Ø*) as in 'have put.' An English verb FST would encode this arrangement as subclasses of stems, as in the more elaborate Russian example.

Classifying stems also allows for a fairly straightforward treatment of exceptionality. For example, the Class 1 noun *soldat* is exceptional in that its genitive plural is not **soldat-ov* as predicted by its pattern, but *soldat-Ø* 'soldier.' This is the genitive plural you expect for a Class 2 noun (see Table 3.1). To handle this we annotate *soldat* lexical entries as Class 1 for all forms *except* the genitive plural, where it is annotated as Class 2. This is shown in (3.13) where a small subset of the lemmas are given.

(3.13) <s o l d a t +noun 1 ^ sg nom>
 <s o l d a t +noun 1 ^ sg gen>
 <s o l d a t +noun 1 ^ pl nom>
 <s o l d a t +noun 2 ^ pl gen>

Another type of exception is represented by *pal'* 'to' overcoat.' What is exceptional about this item is that it does not combine with any inflectional affixes, i.e., it is an indeclinable noun. There are a number of such items in Russian. An FST assigns them their own class and maps all lexical representations to the same affixless surface form, as shown in (3.14).

(3.14) Stem_5:
 <> == \#.

Our last type of exception is what is called a *pluralia tantum* word, such as *scissors* in English, or *trousers*, where there is no morphological singular form. The Russian for 'trousers' is also pluralia tantum: *brjuk-i*. We provide a node in the FST that carries any input string singular features to a node labeled for input

plural features. This is shown in (3.15) as the path <^ sg> inheriting from another path at another node, i.e., <^ pl> at Stem_2. This is because *brjuki* shares plural affixes with other Class 2 nouns.

(3.15) Stem_6:
 <> == \#
 <^ sg> == Stem_2:<^ pl>.

FSTs for lexical analysis are based on I&A style morphology that assumes a straightforward mapping of feature to affix. Affix rivalry of the kind exemplified by Russian upsets this mapping since more than one affix is available for one feature. But by incorporating stem classes and continuation affix classes they can handle such cases and thereby operate over languages with inflectional classes. The subclassification of stems also provides FSM with a way of incorporating exceptional morphological behavior.

3.3.2 Further Remarks on Finite State Lexical Analysis

FSTs can be combined in various ways to encode larger fragments of a language's word structure grammar. Through *union* an FST for, say Russian nouns, can be joined with another FST for Russian verbs. We have already mentioned that FSTs can be cascaded such that the output of FST1 is the input to FST2. This operation is known as *composition*. The FSTs for morphology proper take lemmas as input and give morphologically decomposed string as output. These strings are then the input of morphonological/spelling rules FSTs that are sensitive to morpheme boundaries, i.e., where the symbols ^ and # define contexts for a given rule as we saw with the English plural spelling rule. So, for example, the lemma <k a r t + noun ^ sg nom> maps on to an intermediate level of representation k a r t ^ a #. Another transducer takes this as input path k a r t ^ a # and maps it onto the surface form k a r t a, stripping away the morpheme boundaries and performing any other (morphonological) adjustments. Intermediate levels of representation are dispensed with altogether if the series of transducers is *composed* into a single transducer, as detailed in Roark and Sproat (2007) where <x, y> representing the upper and lower tape of transducer 1 and <y, z> the upper and lower tapes of transducer 2 are composed into a single transducer T as <x, z>, i.e., intermediary <y, z> is implied. As we saw in the previous section, Two-Level Morphology does not compose the morphonological FSTs but intersects them. There is no intermediary level of representation because the FSTs operate orthogonally to a simple finite state automaton representing lexical entries in their lexical (lemma) forms.

Finite state approaches have dominated lexical analysis from Koskenniemi's (1983) implementation of a substantial fragment of Finnish morphology. In the morphology chapters of computational linguistics textbooks the finite state approach takes centre stage, for example, Dale et al. (2000), Mitkov (2004), and Jurafsky and Martin (2007), where it takes center stage for two chapters. In Roark and Sproat (2007) computational morphology *is* FSM. From our demonstration it is not hard to see why this is the case. Implementations of FSTNs are relatively straightforward and extremely efficient, and FSTs provide the simultaneous modeling of morphological generation and analysis. They also have an impressive track record in large-scale multilingual projects, such as the Multext Project (Armstrong 1996) for corpus analysis of many languages including Czech, Bulgarian, Swedish, Slovenian, and Swahili. More recent two-level systems include Ui Dhonnchadha et al. (2003) for Irish, Pretorius and Bosch (2003) for Zulu, and Yona and Wintner (2007) for Hebrew. Finite state morphological environments have been created for users to build their own models, for example, Sproat's (1997) *lex tools* and more recently Beesley and Karttunen's (2003) *xerox finites state tools*. The interested reader should consult the accompanying Web site for this chapter for further details of these environments, as well as DATR style FSTs.

3.4 "Difficult" Morphology and Lexical Analysis

So far we have seen lexical analysis as morphological analysis where there are two assumptions being made about morphologically complex word: (1) one morphosyntactic feature set, such as 'Singular Nominative,'

maps onto one *exponent*, for example, a suffix or a prefix; and (2) the exponent itself is identifiable as a

sequence of symbols lying contiguous to the stem, either on its left (as a prefix) or its right (as a suffix). But in many languages neither (1) nor (2) necessarily hold. As NLP systems are increasingly multilingual, it becomes more and more important to explore the challenges other languages pose for finite state models, which are ideally suited to handle data that conform to assumptions (1) and (2). In this section, we look at various sets of examples that do not conform to a neat I&A analysis. There are sometimes ways around these difficult cases, as we saw with the Russian case where stem classes were used to handle multiple affixes being associated with a single feature. But our discussion will lead to an alternative to I&A analysis that finite state models entail. As we will see in Section 3.5, the alternative W&P approach appears to offer a much more natural account of word structure when it includes the difficult cases.

3.4.1 Isomorphism Problems

It turns out that few languages have a morphological system that can be described as one feature (or feature set) expressed as one morpheme, the exponent of that feature. In other words, *isomorphism* turns out not to be the common situation. In Section 3.3, I carefully chose Turkish to illustrate FSTs for morphology proper because Turkish seems to be isomorphic, a property of agglutinative languages. At the same time derivational morphology tends to be more isomorphic than inflection, hence an English derivational example. But even agglutinative languages can display non-isomorphic behavior in their inflection (3.16) is the past tense set of forms for the verb ‘to be’ (Haspelmath 2002: 33).

(3.16)	ol-i-n	‘I was’
	ol-i-t	‘you (singular) were’
	ol-i	‘he/she was’
	ol-i-mme	‘we were’
	ol-i-tte	‘you (plural) were’
	ol-i-vat	‘they were’

A lexical entry for ‘I was’ would be $\langle o\ l\ +verb\ \wedge\ past\ \wedge\ 1stSg \rangle$ mapping to $o\ l\ \wedge\ i\ \wedge\ n\ \#$. Similarly for ‘we were,’ $\langle o\ l\ +verb\ \wedge\ past\ \wedge\ 1stSPL \rangle$ maps to $o\ l\ \wedge\ i\ \wedge\ mme\ \#$. But what about ‘he/she was’? In this case there is no exponent for the feature set ‘3rd Person Singular’ to map onto; we have lost isomorphism. But in a sense we had already lost it since for all forms in (3.16) we are really mapping a *combination* of features to a single exponent: a Number feature (plural) and a Person feature (1st) map to the single exponent *-mme*, etc. Of course the way out is to use a symbol on the lexical string that describes a feature combination. This is what we did with the Russian examples in Section 3.3.2 to avoid the difficulty, and it is implicit in the Finnish data above. But back to the problem we started with: where there is a ‘missing’ element on the surface string we can use a $\emptyset$ a ‘zero affix,’ for the upper string feature symbol to map to. Indeed, in Tables 3.1 and 3.2 I used $\emptyset$ to represent the morphological complexity of some Russian word-forms. So the Russian FST maps the lemma $\langle z\ a\ k\ o\ n\ +noun\ \wedge\ sg\ \wedge\ nom \rangle$ onto $z\ a\ k\ o\ n\ \wedge\ \emptyset\ \#$. To get rid of the $\emptyset$, a morphonological FST can use empty transitions in the same way it does to delete morpheme boundaries. Variations of this problem can be met with variations of the solution. The French adverb ‘slowly’ is *lentement* where *lent-* is the stem and *-ment* expresses ‘adverb.’ This leaves the middle *-e-* without anything to map onto. The mapping is one upper string symbol to two lower string symbols. The solution is to squeeze in a zero feature, or ‘empty morpheme’ between the stem and the ‘adverb’ feature: $\langle l\ e\ n\ t\ \wedge\ \emptyset\ \wedge\ adverb \rangle$. The converse, two features with a single exponent, is collapsing the two features into a feature set, as we discussed. The alternative is to place zeros on the lower string: $\langle o\ l\ +verb\ Past\ \wedge\ 1stPerson\ \wedge\ Singular \rangle$ maps to $\langle o\ l\ \wedge\ i\ \wedge\ m\ m\ e\ \wedge\ \emptyset\ \#$. But then the choice of what feature has the zero affix is arbitrary.

Finally, we can think of the competition of exponents for a particular feature as a similar kind of isomorphism problem: one feature, several exponents. In Section 3.3.1, we showed how Russian has

inflectional classes, where nouns are grouped according to the choice of suffix they make for a given

TABLE 3.2 Russian Inflectional Classes

	I Zakon	II Karta	III Rukopis'	IV Boloto
Singular				
Nom	<i>zakon-ø</i>	<i>kart-a</i>	<i>rukopis'-ø</i>	<i>bolot-o</i>
Acc	<i>zakon-ø</i>	<i>kart-u</i>	<i>rukopis'-ø</i>	<i>bolot-o</i>
Gen	<i>zakon-a</i>	<i>kart-y</i>	<i>rukopis-i</i>	<i>bolot-a</i>
Dat	<i>zakon-u</i>	<i>kart-e</i>	<i>rukopis-i</i>	<i>bolot-u</i>
Inst	<i>zakon-om</i>	<i>kart-oj</i>	<i>rukopis-ju</i>	<i>bolot-om</i>
Loc	<i>zakon-e</i>	<i>kart-e</i>	<i>rukopis-i</i>	<i>bolot-e</i>
Plural				
Nom	<i>zakon-y</i>	<i>kart-y</i>	<i>rukopis-i</i>	<i>bolot-a</i>
Acc	<i>zakon-y</i>	<i>kart-y</i>	<i>rukopis-i</i>	<i>bolot-a</i>
Gen	<i>zakon-ov</i>	<i>kart-ø</i>	<i>rukopis-ej</i>	<i>bolot-ø</i>
Dat	<i>zakon-am</i>	<i>kart-am</i>	<i>rukopis-jam</i>	<i>bolot-am</i>
Inst	<i>zakon-ami</i>	<i>kart-ami</i>	<i>rukopis-jami</i>	<i>bolot-ami</i>
Loc	<i>zakon-ax</i>	<i>kart-ax</i>	<i>rukopis-jax</i>	<i>bolot-ax</i>

feature. To handle this we used affix classes together with stem classes such that different stem classes were licensed to navigate over different parts of the network where the appropriate affixes are stored.

3.4.2 Contiguity Problems

As well as isomorphism, I&A approaches rely on contiguity: the exponent of a feature should be found at the left or right edge of the stem of the lower string, mirroring the position of the feature relative to the stem on the upper lexical string. But one does not need to look far to find examples of 'noncontiguous' morphology. In Section 3.1, I discussed the potential problem of *sang*, past tense of *sing*. This is an example of a feature mapping onto an exponent, which is really the change we make to the stem's vowel. How can an FST map the correct lower string to the upper string $\langle s \ i \ n \ g \ ^{past} \rangle$? To account for the feature *not* mapping onto an affix, it can use the $\emptyset$ as we did (extensively) with the isomorphism problems above. This then leaves the vowel alternation as the exponent of the feature. One way is to target the stem vowel so that lexical *i* maps onto surface *a*; and allow navigation through the subnetwork just for those items that behave this way (*sing/sang*, *ring/rang*, *spin/span*). This is represented in (3.17). Lexical entries for regular *cook* and irregular *sing* and *ring* are given in (3.18).

(3.17) Ablaut_FST:

```

<$abc> == $abc <>
<s i n g ^ past> == Past_Stem
<r i n g ^ past> == Past_Stem
<^ present> == ^ 0 <>
<^ past> == ^ e d <>
<> == \#.

Past_Stem:
<$abc> == $abc <>
<$vow> == a <>
<^ past> == ^ 0 <>
<> == \#.

```

(3.18) $\langle s \ i \ n \ g \ ^{present} \rangle \ \langle s \ i \ n \ g \ ^{past} \rangle$

<r i n g ^ present> <r i n g ^ past>
<c o o k ^ present> <c o o k ^ past>

The first node provides for the trivial stem string mappings required by regular non-ablauting verbs. The fourth equation expresses lexical ‘present’ mapping to $\emptyset$ (where no account is taken of –s affixation for ‘3rd person singular’). The fifth equation handles regular past formation in –ed, as in *cooked*. But just in case the path to evaluate is <sing ^ past> or <ring ^ past>, an extra node is provided for the evaluation, the node *Past_Stem*. At this node, the symbol class for all stem characters is used to perform the normal upper to lower stem mapping, as in previous DATR examples. The difference is that there is a new symbol class expressed as the variable \$vow, ranging over vowels that alternate with *a* in the past. The path equation <\$vow> == a <> takes the vowel in the stem and maps it to *a*, the vowel alternation that is the exponent of ‘past.’ Any other character belonging to the input (stem) string maps onto itself on the lower string (through <\$abc> == \$abc <>). Finally <^ past> == ^ 0 <> represents the fact that for this class of verbs the zero affix is used as exponent of ‘past.’ The theorem is given in (3.19).

$$\begin{aligned}
 (3.19) \quad & \langle \text{sing} \wedge \text{present} \rangle = \text{sing} \wedge 0 \# . \\
 & \langle \text{sing} \wedge \text{past} \rangle = \text{sang} \wedge 0 \# . \\
 & \langle \text{ring} \wedge \text{present} \rangle = \text{ring} \wedge 0 \# . \\
 & \langle \text{ring} \wedge \text{past} \rangle = \text{rang} \wedge 0 \# . \\
 & \langle \text{cook} \wedge \text{present} \rangle = \text{cook} \wedge 0 \# . \\
 & \langle \text{cook} \wedge \text{past} \rangle = \text{cook} \wedge \text{ed} \# .
 \end{aligned}$$

Another example of noncontiguous morphology is *infixation*, where an affix attaches not to the right or left edges but within the stem itself. (3.20) is an infixation example from Tagalog, a language spoken in the Philippines with nearly sixteen million speakers worldwide (*Ethnologue*). The data are from Mel’čuk (2006: 300).

$$\begin{aligned}
 (3.20) \quad & \text{Sulat ‘write’} \quad \text{Patay ‘kill’} \\
 & \text{ACTIVE} \quad s\text{-}um\text{-}ulat \quad p\text{-}um\text{-}atay \\
 & \text{PASSIVE} \quad s\text{-}in\text{-}ulat \quad p\text{-}in\text{-}atay
 \end{aligned}$$

A way of handling infixation with FSTs is to add an intermediary level of representation, as outlined in Roark and Sproat (2007: 29–31, 39–40). A first transducer maps the location of the affixation to an ‘anchor’ symbol within the stem. A second transducer operates over intermediate representations and maps the anchor with either ^ in ^ or ^ um ^ depending on the voice feature (Active or Passive). Note that features need to be preserved from the lexical to intermediate levels. This is because the only difference between the two string types is the anchor; and as it is an affix it needs boundaries, and as it is an infix the boundaries need to be on both left and right. This is important in case there are any morphonological consequences to the infixation. This approach informs the DATR transducers in (3.21) and (3.23).

$$\begin{aligned}
 (3.21) \quad & \text{Intermediate\_FST:} \\
 & \langle \$features \rangle == \$features \langle \rangle \\
 & \langle \$abc \rangle == \$abc \langle \rangle \\
 & \langle ^ \$abc \$vow \rangle == ^ \$abc \& \$vow \langle \rangle \\
 & \langle \rangle == .
 \end{aligned}$$

$$\begin{aligned}
 (3.22) \quad & \langle \text{active} \wedge s \text{ ul a t} \rangle \quad \langle \text{passive} \wedge s \text{ ul a t} \rangle \\
 & \langle \text{active} \wedge p \text{ a t a y} \rangle \quad \langle \text{passive} \wedge p \text{ a t a y} \rangle
 \end{aligned}$$

As feature annotations are needed in the output string, we need a symbol class for them, expressed as the variable \$features; then identity mapping is treated as in the previous FSTs. The second equation is the (by now) familiar trivial stem character mapping. Note that lexical entries come with features to the left of the canonical stem (3.22). The third equation in (3.21) represents the way the FST targets the first character

followed by the first vowel of a stem. The right hand side expresses the transduction: the boundary symbol

is preserved, as is the first letter of the stem and the first vowel of the stem. But at the same time a symbol & is inserted between them. This is our anchor. The output of the transducer is therefore: active ^ s & u l a t, and passive ^ s & u l a t. These strings are then the input to a second transducer in (3.23) by recasting them as paths.

(3.23) Infixation_FST:
 <active ^> == Active:<>
 <passive ^> == Passive:<>.
 Active:
 <\$abc> == \$abc <>
 <& > == ^ um ^ <>
 <> == \#.
 Passive:
 <\$abc> == \$abc <>
 <& > == ^ in ^ <>
 <> == \#.

Here the feature symbols are used to decode the anchor &. Paths beginning <active ^> are evaluated at the Active node, and paths beginning <passive ^> at the Passive node. The first line at each node maps stem letters to themselves in the normal way. The second line maps & to a boundary delimited um at the Active node and in at the Passive node. End of word boundaries are inserted when there is no more path to extend, as previously. The theorem is given in (3.24).

(3.24) <active ^ s & u l a t> = s ^ um ^ u l a t #
 <passive ^ s & u l a t> = s ^ in ^ u l a t #

Our final example of noncontiguous morphology is where the root is interrupted, and at the same time so is the exponent: “where affixes interrupt roots and are interrupted by elements of roots themselves.” (Mel’čuk 2006: 300). Mel’čuk uses the term ‘transfixation’ but in the FST literature this type is usually called ‘root and template,’ ‘root and pattern,’ or simply ‘non-concatenative’ morphology. In Arabic, the root for ‘draw’ is the consonant template, or skeleton, *r-s-m*. The exponent is the flesh. For ‘active perfect,’ the exponent is the vowel series *-a-a-*, for ‘passive perfect,’ there is another exponent *-u-i-*. So *rasam(a)* ‘he has drawn,’ and *rusim(a)* ‘it has been drawn’ (where the bracketed *-a* is the more normal suffix exponent): the root is interrupter and interrupted, as is the exponent. Another way of thinking about situations of this sort is to have different *tiers* of structural information. Booij (2007: 37) gives a Modern Hebrew example. The root of the verb *grow* is the consonant series *g-d-l*. The root occupies one tier. The formal distinction between the word ‘grow’ *gadal* and its nominalization ‘growth’ *gdila* is expressed by a pattern of consonants and vowels occupying separate tiers. This is represented in (3.25).

(3.25) gadal
 Tier 1 a a
 Tier 2 C V C V C
 Tier 3 g d l
 gdila
 Tier 1 i a
 Tier 2 C C V C V
 Tier 3 g d l

There are therefore three separate pieces of surface information: the root (consonants) as tier 3 information, the exponent (vowels) as tier 1, and the instruction for how they interleave as tier 2. How is this modeled as finite state machine? A method reported in Kiraz (2000) for Arabic and widely discussed in the FSM literature is to have an n -tape FST where $n > 1$. Each tier has representation on one of

several lower tapes. Another tape is also provided for concomitant prefixation or suffixation. Rather ingeniously, a noncontiguous problem is turned into a contiguous one so that it can receive a contiguous solution. Roark and Sproat (2007) propose a family of transducers for different CV patterns (where V is specified) and the morphosyntactic information they express. The union of these transducers is composed with a transducer that maps the root consonants to the Cs. The intermediate level involving the pattern disappears.

3.5 Paradigm-Based Lexical Analysis

The various morphological forms of Russian *karta* in (3.9) were presented in such a way that we could associate form and meaning difference among the word-forms by consulting the cell in the table, the place where case and number information intersect with word-form. The presentation of a lexeme's word-forms as a *paradigm* provides an alternative way of capturing word structure that does not rely on either isomorphism or contiguity. For this reason, the W&P approach has been adopted by the main stream of morphological theorists with the view that "paradigms are essential to the very definition of a language's inflectional system" (Stewart and Stump 2007: 386). A suitable representation language that has been used extensively for paradigm-based morphology is the lexical knowledge representation language DATR, which up until now we have used to demonstrate finite state models. In this section, we will outline some of the advantages of paradigm-based morphology. To do this we will need to slightly extend our discussion of the DATR formalism to incorporate the idea of *inheritance* and *defaults*.

3.5.1 Paradigmatic Relations and Generalization

The FSM demonstrations above have been used to capture properties not only about single lexical items but whole classes of items. In this way, lexical analysis goes beyond simple listing and attempts generalizations. It may be helpful at this point to summarize how generalizations have been captured. Using symbol classes, FSTs can assign stems and affixes to categories, and encode operations over these categories. In this way, they capture classes of environments and changes for morphonological rules, and morpheme orderings that hold for classes of items, as well as selections when there is a choice of affixes for a given feature. But there are other generalizations that are properties of paradigmatic organization itself, what we could think of as *paradigmatic relations*. To illustrate let us look again at the Russian inflectional class paradigms introduced earlier in Table 3.1, and presented again here as Table 3.3.

TABLE 3.3 Russian Inflectional Classes

	I Zakon	II Karta	III Rukopis'	IV Boloto
	Singular			
Nom	<i>zakon-ø</i>	<i>kart-a</i>	<i>rukopis'-ø</i>	<i>bolot-o</i>
Acc	<i>zakon-ø</i>	<i>kart-u</i>	<i>rukopis'-ø</i>	<i>bolot-o</i>
Gen	<i>zakon-a</i>	<i>kart-y</i>	<i>rukopis-i</i>	<i>bolot-a</i>
Dat	<i>zakon-u</i>	<i>kart-e</i>	<i>rukopis-i</i>	<i>bolot-u</i>
Inst	<i>zakon-om</i>	<i>kart-oj</i>	<i>rukopis-ju</i>	<i>bolot-om</i>
Loc	<i>zakon-e</i>	<i>kart-e</i>	<i>rukopis-i</i>	<i>bolot-e</i>
	Plural			
Nom	<i>zakon-y</i>	<i>kart-y</i>	<i>rukopis-i</i>	<i>bolot-a</i>
Acc	<i>zakon-y</i>	<i>kart-y</i>	<i>rukopis-i</i>	<i>bolot-a</i>
Gen	<i>zakon-ov</i>	<i>kart-ø</i>	<i>rukopis-ej</i>	<i>bolot-ø</i>
Dat	<i>zakon-am</i>	<i>kart-am</i>	<i>rukopis-jam</i>	<i>bolot-am</i>

Inst	<i>zakon-ami</i>	<i>kart-ami</i>	<i>rúkopis-jami</i>	<i>bolot-ami</i>
Loc	<i>zakon-ax</i>	<i>kart-ax</i>	<i>rukopis-jax</i>	<i>bolot-ax</i>

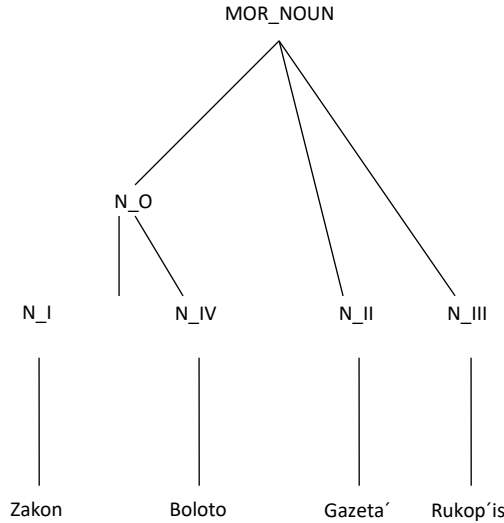


FIGURE 3.4 Russian nouns classes as an inheritance hierarchy (based on Corbett, C.G. and Fraser, N.M., *Network Morphology: A DATR account of Russian inflectional morphology*. In Katamba, F.X. (ed), pp. 364–396. 2003).

One might expect that each class would have a unique set of forms to set them apart from the other class. So looking horizontally across a particular cell pausing at, say, the intersection of Instrumental and Singular, there would be four different values, i.e., four different ways of forming a Russian singular instrumental noun. Rather surprisingly, this does not happen here: for Class I the suffix *-om* is used, for Class II *-oj*, for Class III *-ju*, but Class IV does the same as Class I. Even more surprising is that there is not a single cell where a four-way distinction is made. Another expectation is that within a class, each cell would be different from the other, so, for example, forming a nominative singular is different from a nominative plural. While there is a tendency for *vertical* distinctions across cells, it is only a tendency. So for Class II, dative singular is in *-e*, but so is locative singular. In fact, in the world's languages it is very rare to see fully horizontal and fully vertical distinctions. Recent work by Corbett explores what he calls 'canonical inflectional classes' and shows that the departures from the canon are the norm, so canonicity does not correlate with frequency (Corbett 2009). Paradigm-based lexical analysis takes the departures from the canon as the starting point. It then attempts to capture departures by treating them as *horizontal relations* and *vertical relations*, as well as a combination of the two. So an identical instrumental singular for Classes I and IV is a relation between these classes at the level of this cell. And in Class II there is a relationship between the dative and locative singular. To capture these and other paradigmatic relations in Russian, the inflectional classes in Table 3.2 can be given an alternative presentation as a hierarchy of nodes down which are inherited instructions for forming morphologically complex words (Figure 3.4).

Horizontal relations are expressed as inheritance of two daughter nodes from a mother node. The node *N_O* stores the fact about the instrumental singular, and both Classes I and IV, represented as *N_I* and *N_IV*, inherit it. They also inherit genitive singular, dative singular, and locative singular. This captures the insight of Russian linguists that these two classes are really versions of each other, for example, Timberlake (2004: 132–141) labels them 1a and 1b. Consulting Table 3.2, we see that all classes in the plural form the dative, instrumental, and locative in the same way. The way these are formed should therefore be stated at the root node *MOR_NOUN* and from there inherited by all classes. In the hierarchy, leaf nodes are the lexical entries themselves, each a daughter of an appropriate inflectional class node.

The DATR representation of the lexical entry `Karṭa` and the node form that it inherits is given in (3.25). The ellipsis `' '` indicates here and elsewhere that the node contains more equations, and is not part of the DATR language.

```

(3.25)  Karta:
        <mor> == N_II
        <stem> == kart.
N_II:
        <mor sg nom> == "<stem>" ^a
        <mor sg acc> == "<stem>" ^u
        <mor sg dat> == "<stem>" ^e
        <mor sg loc> == "<stem>" ^e
        ...

```

The first equation expresses that the *Karta* node inherits path value equations from the *N_II* node. Recall that in DATR a subpath implies any extension of itself. So the path *<mor>* implies any path that is an extension of *<mor>* including *<mor sg nom>*, *<mor sg acc>*, *<mor sg dat>*, *<mor sg loc>*, etc. These are all paths that are specified with values at *N_II*. So the first equation in (3.25) is equivalent to the equations in (3.26). But we want these facts to be inherited by *Karta* rather than being explicitly stated at *Karta* to capture the fact that they are shared by other (Class II) nouns.

```

(3.26)  Karta:
        <mor sg nom> == "<stem>" ^a
        <mor sg acc> == "<stem>" ^u
        <mor sg dat> == "<stem>" ^e
        <mor sg loc> == "<stem>" ^e
        ...

```

The value of these paths at *N_II* is complex: the concatenation of the value of another path and an exponent. The value of the other path is the string expressing a lexical entry's stem. In fact, this represents how paradigm-based approaches model word structure: the formal *realization* of a set of morphosyntactic features by a rule operating over stems. The value of "*<stem>*" depends on what lexical entry is the object of the morphological query. If it is *Karta*, then it is the value for the path *<stem>* at the node *Karta* (3.25). The quoted path notation means that inheritance is set to the context of the initial query, here the node *Karta*, and is not altered even if the evaluation of *<stem>* is possible locally. So we could imagine a node (3.27) similar to (3.25) *N_II* but adding an eccentric local fact about itself, namely that Class II nouns always have *zakon* as their stem, no matter what their real stem is.

```

(3.27)  N_II:
        <mor sg nom> == "<stem>" ^a
        <stem> == zakon
        ...

```

Nonetheless, the value of *<mor sg nom>* will not involve altering the initial context of *<stem>* to the local context. By quoting *<stem>* the value will be *kart^a* and not *zakon^a*. There are equally occasions where local inheritance of paths is precisely what is needed. Equation 3.25 fails to capture the vertical relation that for Class II nouns the dative singular and the locative singular are the same. Local inheritance expresses this relation, shown in (3.28).

```

(3.28)  N_II:
        <mor sg dat> == "<stem>" ^e
        <mor sg loc> == <mor sg dat>
        ...

```

The path `<mor sg loc>` locally inherits from `<mor sg dat>`, so that both paths have the value `karte` where `Karta` is the query lexical entry.

Horizontal relations are captured by hierarchical relations. (3.29) is the partial hierarchy involving MOR_NOUN, N_O, N_I and N_IV.

```
(3.29)  MOR_NOUN:
         <mor pl loc> == "<stem>" ax
         <mor sg dat> == "<stem>" e
         ...

         N_O:
         <> == MOR_NOUN
         <mor sg gen> == "<stem>" a
         <mor sg dat> == "<stem>" u
         <mor sg inst > == "<stem>" om
         <mor sg loc > == "<stem>" e.

         N_IV:
         <> == N_O
         <mor sg nom > == "<stem>" o
         <mor pl nom > == "<stem>" a
         ...

         N_I:
         <> == N_O
         <mor sg nom > == "<stem>"
         <mor pl nom > == "<stem>" y
         ...
```

Facts about cells that are shared by inflectional classes are stated as path: value equations gathered at MOR_NOUN. These include instructions for forming the locative singular and the locative plural. Facts that are shared only by Classes I and IV are stored at an intermediary node N_O: the genitive, dative, instrumental, and locative singular. Nodes for Classes I and IV inherit from this node, and via this node they inherit the wider generalizations stated at the hierarchy's root node. The passing of information down the hierarchy is through the empty path <>, which is the leading subpath of every path that is not defined at the local node. So at N_I the empty path implies <mor gen sg> at its mother node N_O but *not* <mor sg nom> because this path is already defined at N_I.

From Figure 3.4, we can observe a special type of relation that is at once vertical and horizontal. In the plural, the accusative is the same as the nominative in Class I, but this same vertical relation extends to all the other classes: they all have an accusative/nominative identity in the plural. To capture this we store the vertical relation at the root node so that all inflectional class nodes can inherit it (3.30).

```
(3.30)  MOR_NOUN:
         <mor pl acc > == "<mor pl nom >"
         ...
```

It needs to be clear from (3.30) that what is being hierarchically inherited is not an exponent of a feature set but a way of getting the exponent. The quoted path at the right hand side expresses that the nominative plural value depends on the global context, i.e., what noun is being queried: if it is a Class I noun it will

be stem plus *-y*, and if it is a Class IV noun it will be stem plus *-a*. These will therefore be the respective values for Class I and Class IV accusative singular forms. In other words, the horizontal relation being captured is the vertical relation that the accusative and nominative plural for a given class is the same, although in principle the value itself may be different for each class.

3.5.2 The Role of Defaults

Interestingly the identity of accusative with nominative is not restricted to the plural. From Figure 3.4 we see that for Classes I, III, and IV the singular accusative is identical to the nominative but Class II has separate exponents. If inheritance from `MOR_NOUN` is interpreted as default inheritance then (3.31) captures a strong tendency for the accusative and nominative to be the same. The bound variable `$num` ranges over `sg` and `pl` atoms.

```
(3.31)  MOR_NOUN
        <mor $num acc > == "<mor $num nom>"
        ...
```

This vertical relation does not, however, extend to Class II which has a unique accusative singular exponent. Class II inherits facts from `MOR_NOUN`, as any other class does. But it needs to override the fact about the accusative singular (3.32).

```
(3.32)  N_II:
        <> == MOR_NOUN
        <mor sg nom> == "<stem>" a
        <mor sg acc> == "<stem>" u
        ...
```

A path implies any extension of itself, but the value associated with the implied path is by default and can be overridden by an explicitly stated extended path. In (3.32) the empty path implies all of its extensions and their values held at `MOR_NOUN`, including `<mor sg acc> == <mor sg nom>`. But because `<mor sg acc>` is stated locally at `N_II`, the explicitly stated local evaluation overrides the (implied) inherited one. In similar vein we can capture the locative singular in *-e* as a generalization over all classes *except* Class III (see Table 3.2) by stating `<mor sg loc> == "<stem>" e` at the root node, but also stating `<mor sg loc> == "<stem>" i` at the node `N_III`.

Defaults also allow for a straightforward account of exceptional or semi-regular lexical entries. In Section 3.3.1, I gave several examples from Russian to show how exceptionality could be captured in FSM. Let us briefly consider their treatment in a paradigm-based approach. Recall that the item *soldat* ‘soldier’ is a regular Class I noun in every respect except for the way it forms its genitive plural. Instead of **soldatov* it is simply *soldat*. As in the FSM account, it can be assigned Class II status just for its genitive plural by overriding the default for its class, and inheriting the default from another class.

```
(3.33)  Soldat:
        <mor> == N_I
        <stem> == soldat
        <mor pl gen> == N_II.
```

The other example was a pluralia tantum noun, *brjuki* ‘trousers.’ Here the singular had plural morphology. This is simply a matter of overriding the inheritance of `<mor sg>` paths from `N_II` with the equation `<mor sg> == <mor pl>`. Of course `<mor pl>` and its extensions will be inherited from the mother node in the same way it is for all other Class II nouns.

```
(3.34)  Brjuki:
        <mor> == N_II
        <stem> == brjuk
        <mor sg> == <mor pl>.
```

Finally, indeclinable *pal’to* can be specified as inheriting from a fifth class that generalizes over all

indeclinables. It is not *pal'to* that does the overriding but the class itself. All extensions of <mor> are overridden and assigned the value of the lexical entry's stem alone, with no exponent.

(3.35) N_V :
 $\langle \rangle == MOR\_NOUN$
 $\langle mor \rangle == "<stem>"$.

3.5.3 Paradigm-Based Accounts of Difficult Morphology

In Section 3.4, we described ‘difficult’ morphology as instances where isomorphism, one feature (set) mapping to one exponent, breaks down. The Russian demonstration of the paradigm-based approach has run into this almost without noticing. We have already seen how more than one feature can map onto a single exponent. First, an exponent is a value associated with an attribute path that defines *sets* of features: Number and Case in our running example of Russian. Second, the whole idea behind vertical relations is that two different feature sets can map onto a single exponent: the *-e* suffix maps onto two feature sets for Class II nouns, the singular dative and the singular locative, similarly the accusative and nominative mapping onto a single exponent for all but Class II. This is handled by setting one attribute path to inherit from another attribute path. In paradigm-based morphology, this is known as a *rule of referral* (Stump 1993, 2001: 52–57). Then the reverse problem of more than exponent realizing a single feature (set) is handled through the stipulation of inflectional classes, so that affixes for the same feature are not in competition. Finally, a feature that does not map onto any exponent we treat as *zero exponence* rather than a zero affix. For example, the genitive plural of a Russian Class II noun is the stem plus nothing. The attribute path is associated with the query lexical entry’s stem, and nothing more.

(3.36) N_II
 $\langle \rangle == MOR\_NOUN$
 $\langle mor\ pl\ gen \rangle == "<stem>"$
 ...

In paradigm-based accounts, a stem of a word in a given cell of a particular set of features is in contrast with stems of the same word in different cells. Morphological structure is then the computation of what makes the stem contrastive together with the feature content of the cell. For this reason, affixation is not given any special status: contrast could be through ablaut, stress shift, or zero exponence as shown above since it also can mark a contrast: by having no exponence, the genitive plural is opposed to all other cells since they do have an exponent. Noncontiguous morphology does not pose a problem for paradigm-based approaches, as Stump (2001) demonstrates. The exponent a rule introduces does not have to target stem edges (affixation) but can target any part of the stem: for example, *sing* has the past tense form *sang*. Cahill and Gazdar (1999) handle ablaut morphological operations of this kind in German plural nouns by defining stems as syllable sequences where the syllable itself has internal structure: an onset consonant, vowel, and a coda consonant. The target of the rule is definable as stem-internal, the syllable vowel. With an inheritance hierarchy, ablaut operations are captured as semi-regular by drawing together items with the similar ablaut patterns under a node that houses the non-concatenative rule: which part of the syllable is modified, and how it is modified. Many nouns undergo ablaut and simultaneously use regular suffixation, for example, *Mann* ‘man’ has plural *Männer*. The hierarchical arrangement allows for a less regular ablaut together with inheritance of the more regular suffixation rule, *-er* added to the right edge of stem, i.e., after the coda of rightmost syllable. Default inheritance coupled with realization rules simultaneously captures multiple exponence, semi-regularity, and nonlinear morphology.

In more recent work, Cahill (2007) used the same ‘syllable-based morphology’ approach for root-and-pattern morphological description that we discussed in Section 3.4 with reference to Hebrew. The claim is that with inheritance hierarchies, defaults and the syllable as a unit of description, Arabic verb morphology lies more or less within the same problem (and solution) space as German (and English) ablaut cases. A morphologically complex verb such as *kutib* has

the default structural description: <mor word> == "<agr prefix>" "<tense prefix>" "<ph root form>" "<agr suffix>". Each quoted path expresses the value of an exponent

inferable from the organization of the network. These exponents can appear as prefixes and suffixes, which are by default null. But an exponent can be a systematic internal modification of the root: “<ph root form>”. Just as for German, the root (or stem) is syllable defined, and more specifically as a series of consonants, e.g., *ktb* ‘write.’ In syllable terms, these are (by default) the onset of the first (logical order) syllable and the onset and coda of the second. The vowels that occupy nucleus positions of the two syllables can vary and these variations are exponents of tense and mood. So one possible perfect passive form is *kutib* that contrasts with perfect passive *katab*. There are many such root alternations and associated morphosyntactic property sets. To capture similarities and differences, they are organized into default inheritance hierarchies.

This is necessarily an oversimplified characterization of the complexity that Cahill’s account captures, but the important point is that defaults and W&P morphology can combine to give elegant accounts of noncontiguous morphology. In Soudi et al. (2007), the collection of computational Arabic morphology papers where Cahill’s work appears, it is significant that most of the symbolic accounts are lexeme-based and make use of inheritance; two are DATR theories. Finkel and Stump (2007) is another root-and-pattern account, this time of Hebrew. It also is W&P morphology with default inheritance hierarchies. Its implementation is in KATR, a DATR with ‘enhancements,’ including a mechanism for expressing morphosyntactic features as unordered members of sets as well as ordered lists to better generalize the Hebrew data. Kiraz (2008) and Sproat (2007) note that Arabic computational morphology has been neglected; this is because “root-and-pattern morphology defies a straightforward implementation in terms of morpheme concatenation” (Sproat 2007: vii). Cahill (2007), and Finkel and Stump (2007) offer an alternative W&P approach, which suggests that perhaps Arabic is not really “specifically engineered to maximize the difficulties for automatic processing.”

3.5.4 Further Remarks on Paradigm-Based Approaches

Many computational models of paradigm-based morphological analysis are represented in the DATR lexical knowledge representation language. These include analyses of major languages, for example, Russian in Corbett and Fraser (2003), the paper which the W&P demonstration in this chapter is based on, and more recently in Brown and Hippisley (forthcoming); also Spanish (Moreno-Sandoval and Goñi-Menoyo 2002), Arabic as we have seen (Cahill 2007), German (Cahill and Gazdar 1999, Kilbury 2001), Polish (Brown 1998), as well as lesser known languages, for example, Dalabon (Evans et al. 2000), Mayali (Evans et al. 2002), Dhaasanac (Baerman et al. 2005: Chapter 5), and Arapesh (Fraser and Corbett 1997). The paradigm-based theory Network Morphology (Corbett and Fraser 2003, Brown and Hippisley forthcoming) is formalized in DATR. The most well articulated paradigm-based theory is Paradigm Function Morphology (Stump 2001, 2006), and DATR has also been used to represent Paradigm Function Morphology descriptions (Gazdar 1992). DATR’s mechanism for encoding default inference is central to Network Morphology and Paradigm Function Morphology. Defaults are used in other theoretical work on the lexicon as part of the overall system of language. For example, HPSG (Pollard and Sag 1994, Sag et al. 2003), uses inheritance hierarchies to capture shared information; inheritance *by default* is used for specifically lexical descriptions in some HPSG descriptions, for example, Krieger and Nerbonne (1992) and more recently Bonami and Boyé (2006).*

We close this section with a comment about DATR. Throughout our discussion of FSM and the paradigm-based alternative, we have used DATR as the demonstration language. But it is important to note that DATR theories are ‘one way’ as they start with the lexical representation and generate the

* HPSG has been ambivalent over the incorporation of defaults for lexical information but Bonami and Boyé (2006) are quite clear about its importance:

In the absence of an explicit alternative, we take it that the use of defaults is the only known way to model regularity in an HPSG implementation of the stem space.

The latest HPSG textbook appears to endorse defaults for the lexicon (Sag et al. 2003: 229–236).

surface representation; they do not start with the surface representation and parse out the lexical string. In and of itself, this restriction has not hampered a discussion of the major lexical analysis issues, which is the motivation for using DATR in this chapter but it is important to bear in mind this aspect of DATR. There are ways round this practical restriction. For example, a DATR theory could be compiled into a database so that its theorem is presented as a surface string to lexical string look up table. Earlier work has experimented with extending DATR implementations to provide inference in reverse. Langer (1994) proposes an implementation of DATR that allows for ‘reverse queries,’ or inference operating in reverse. Standardly, the query is a specific node/path/value combination, for example, *Karta*:<mor nom sg>. The value is what is inferred by this combination. By treating a DATR description as being analogous to a context free phrase structure grammar (CF-PSG), with left hand sides as nonterminal symbols, which rewrite as right hand sides that include nonterminal and terminal symbols, reverse query can be tackled as a CF-PSG bottom-up parsing problem. You start with the value string (*kart^a*) and discover how it has been inferred (*Karta*:<mor nom sg>). Finally, a DATR description could be used to generate the pairing of lexical string to surface string, and these pairings could then be the input to an FST inducer to perform analysis. For example, Gildea and Jurafsky (1995, 1996) use an augmented version of Oncina et al.’s (1993) OSTIA algorithm (Onward Subsequential Transducer Inference Algorithm) to induce phonological rule FSTs. More recently, Carlson (2005) reports on promising results of an experiment inducing FSTs from paradigms of Finnish inflectional morphology in Prolog.

3.6 Concluding Remarks

If the building blocks of natural language texts are words, then words are important units of information, and language-based applications should include some mechanism for registering their structural properties. Finite state techniques have long been used to provide such a mechanism because of their computational efficiency, and their invertibility: they can be used both to generate morphologically complex forms from underlying representations, and parse morphologically complex forms into underlying representations. The linguistic capital of the FSM is an I&A model of word structure. Though many languages, including English, contain morphologically complex objects that resist an I&A analysis, FSM handles these situations by recasting these problems as I&A problems: isomorphism is retained through empty transitions, collecting features into symbol sets, and introducing stem classes and affix classes. For nonlinear morphology, infixation, and root-and-template, the problem is recast as a linear one and addressed accordingly. FSM can capture morphological generalization, exceptionality, and the organization of complex words into inflectional classes. An alternative to FSM is an approach based on paradigmatic organization where word structure is computed by the stem’s place in a cell in a paradigm, the unique clustering of morphologically meaningful elements, and some phonologically contrasting element, not necessarily an affix, and feasibly nothing. W&P approaches appear to offer a better account of what I have called difficult morphology. They also get at the heart of morphological generalization.

Both approaches to lexical analysis are strongly rule based. This puts lexical analysis at odds with most fields of NLP, including computational syntax where statistics plays an increasingly important role. But Roark and Sproat (2007: 116–117) observe that hand-written rules can take you a long way in a morphologically complex language; at the same time ambiguity does not play such a major role in morphological analysis as it does in statistical analysis where there can be very many candidate parse trees for a single surface sentence. That is not to say that ambiguity is not a problem in lexical analysis: given the prominence we have attached to inflectional classes in this chapter, it is not hard to find situations where a surface string has more than one structure. This is the case for all vertical relations, discussed in the previous section. In Russian, the string *karte* can be parsed as a dative or locative, for example. A worse case is *rukopisi*. Consulting Table 3.2 you can see that this form could be a genitive, dative, or locative singular; or nominative or accusative plural. Roark and Sproat go on to note that resolving these

ambiguities requires too broad a context for probability information to be meaningful. That is not to say

that morphological analysis does not benefit from statistical methods. Roark and Sproat give as examples Goldsmith (2001), a method for inducing from corpora morphonological alternations of a given lemma (Goldsmith 2001), and Yarowski and Wicentowski (2001), who use pairings of morphological variants to induce morphological analyzers.

It should be noted that the important place of lexical/morphological analysis in a text-based NLP system is not without question. In IR, there is a view that symbolic, rule-based models are difficult to accommodate within a strongly statistically oriented system, and the symbolic statistical disconnect is hard to resolve. Furthermore, there is evidence that morphological analysis does not greatly improve performance: indeed stemming can lower precision rates (Tzoukerman et al. 2003). A rather strong position is taken in Church (2005), which amounts to leaving out morphological analysis altogether. His paper catalogues the IR community's repeated disappointments with morphological analysis packages, primarily due to the fact that morphological relatedness does not always equate with semantic relatedness: for example, *awful* has nothing to do with *awe*. He concludes that simple listing is preferable to attempting lexical analysis. One response is that inflectional morphologically complex words are more transparent than derivational, so less likely to be semantically disconnected from related forms. Another is that Church's focus is English which is morphologically poor anyway, whereas with other major languages such as Russian, Arabic, and Spanish listing may not be complete, and will certainly be highly redundant.

Lexical analysis is in fact increasingly important as NLP reaches beyond English to other languages, many of which have rich morphological systems. The main lexical analysis model is FSM that has a good track record for large-scale systems, English as well as multilingual. The paradigm-based model is favored by linguists as an elegant way of describing morphologically complex objects. Languages like DATR can provide a computable lexical knowledge representation of paradigm-based theories. Communities working within the two frameworks can benefit from one another. Kay (2004) observes that early language-based systems were deliberately based on scientific principles, i.e., linguistic theory. At the same time, giving theoretical claims some computational robustness led to advances in linguistic theory. The fact that many DATR theories choose to implement the morphonological variation component of their descriptions as FSTs shows the intrinsic value of FSM to morphology as a whole. The fact that there is a growing awareness of the paradigm-based model within the FSM community, for example, Roark and Sproat (2007) and Karttunen (2003) have implementations of Paradigm Function Morphology accounts in FSTs, may lead to an increasing awareness of the role paradigm relations play in morphological analysis, which may lead to enhancements in conventional FST lexical analysis. Langer (1994) gives two measures of adequacy of lexical representation: declarative expressiveness and accessing strategy. While a DATR formalized W&P theory delivers on the first, FSM by virtue of generation and parsing scores well on the second. Lexical analysis can only benefit with high scores in both.

Acknowledgments

I am extremely grateful to Roger Evans and Gerald Gazdar for their excellent comments on an earlier draft of the chapter which I have tried to take full advantage of. Any errors I take full responsibility for. I would also like to thank Nitin Indurkha for his commitment to this project and his gentle encouragement throughout the process.

References

- Armstrong, S. 1996. Multext: Multilingual text tools and corpora. In: Feldweg, H. and Hinrichs, W. (eds). *Lexikon und Text*. Tübingen, Germany: Niemeyer.

Aronoff, M. and Fudeman, K. 2005. *What is Morphology?* Oxford, U.K.: Blackwell.

- Arpe, A. et al. (eds). 2005. *Inquiries into Words, Constraints and Contexts (Festschrift in Honour of Kimmo Koskeniemi on his 60th Birthday)*. Saarijärvi, Finland: Gummerus.
- Baerman, M., Brown, D., and Corbett, G.G. 2005. *The Syntax-Morphology Interface: A Study of Syncretism*. Cambridge, U.K.: Cambridge University Press.
- Beesley, K. and Karttunen, L. 2003. *Finite State Morphology*. Stanford, CA: CSLI.
- Bonami, O. and Boyé, G. 2006. Deriving inflectional irregularity. In: Müller, S. (ed). *Proceedings of the HPSG06 Conference*. Stanford, CA: CSLI.
- Booij, G. 2007. *The Grammar of Words*. Oxford, U.K.: Oxford University Press.
- Brown, D. and Hippisley, A. Under review. *Default Morphology*. Cambridge: Cambridge University Press.
- Brown, D. 1998. Defining 'subgender': Virile and devirilized nouns in Polish. *Lingua* 104 (3–4). 187–233.
- Cahill, L. 2007. A syllable-based account of Arabic morphology. In: Soudi, A. (eds). pp. 45–66.
- Cahill, L. and Gazdar, G. 1999. German noun inflection. *Journal of Linguistics* 35 (1). 1–42.
- Carlson, L. 2005. Inducing a morphological transducer from inflectional paradigms. In: Arpe et al. (eds). pp. 18–24.
- Chomsky, N. and Halle, M. 1968. *The Sound Pattern of English*. New York: Harper & Row.
- Church, K.W. 2005. The DDI approach to morphology. In Arpe et al. (eds). pp. 25–34.
- Corbett, G.G. 2009. Canonical inflection classes. In: Montermini, F., Boyé, G. and Tseng, J. (eds). *Selected Proceedings of the Sixth Decembrettes: Morphology in Bordeaux*, Somerville, MA: Cascadia Proceedings Project. www.lingref.com, document #2231, pp. 1–11.
- Corbett, G.G. and Fraser, N.M. 2003. Network morphology: A DATR account of Russian inflectional morphology. In: Katamba, F.X. (ed). pp. 364–396.
- Dale, R., Moisl, H., and Somers, H. (eds). 2000. *Handbook of Natural Language Processing*. New York: Marcel Dekker.
- Evans, N., Brown, D., and Corbett, G.G. 2000. Dalabon pronominal prefixes and the typology of syncretism: A network morphology analysis. In: Booij, G. and van Marle, J. (eds). *Yearbook of Morphology 2000*. Dordrecht, the Netherlands: Kluwer. pp. 187–231.
- Evans, N., Brown, D., and Corbett, G.G. 2002. The semantics of gender in Mayali: Partially parallel systems and formal implementation. *Language* 78 (1). 111–155.
- Evans, R. and Gazdar, G. 1996. DATR: A language for lexical knowledge representation. *Computational Linguistics* 22. 167–216.
- Finkel, R. and Stump, G. 2007. A default inheritance hierarchy for computing Hebrew verb morphology. *Literary and Linguistics Computing* 22 (2). 117–136.
- Fraser, N. and Corbett, G.G. 1997. Defaults in Arapesh. *Lingua* 103 (1). 25–57.
- Gazdar, G. 1985. Review article: Finite State Morphology. *Linguistics* 23. 597–607.
- Gazdar, G. 1992. Paradigm-function morphology in DATR. In: Cahill, L. and Coates, R. (eds). *Sussex Papers in General and Computational Linguistics*. Brighton: University of Sussex, CSRP 239. pp. 43–53.
- Gibbon, D. 1987. Finite state processing of tone systems. *Proceedings of the Third Conference, European ACL*, Morristown, NJ: ACL. pp. 291–297.
- Gibbon, D. 1989. 'tones.dtr'. Located at <ftp://ftp.informatics.sussex.ac.uk/pub/nlp/DATR/dtrfiles/tones.dtr>
- Gildea, D. and Jurafsky, D. 1995. Automatic induction of finite state transducers for single phonological rules. *ACL* 33. 95–102.

- Gildea, D. and Jurafsky, D. 1996. Learning bias and phonological rule induction. *Computational Linguistics* 22 (4). 497–530.
- Goldsmith, J. 2001. Unsupervised acquisition of the morphology of a natural language. *Computational Linguistics* 27 (2). 153–198.
- Haspelmath, M. 2002. *Understanding Morphology*. Oxford, U.K.: Oxford University Press.
- Hockett, C. 1958. Two models of grammatical description. In: Joos, M. (ed). *Readings in Linguistics*. Chicago, IL: University of Chicago Press.

- Jurafsky, D. and Martin, J.H. 2007. *Speech and Language Processing*. Upper Saddle River, NJ: Pearson/Prentice Hall.
- Kaplan, R.M. and Kay, M. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20. 331–378.
- Karttunen, L. 2003. Computing with realizational morphology. In: Gelbukh, A. (ed). *CICLing 2003 Lecture Notes in Computer Science 2588*. Berlin, Germany: Springer-Verlag. pp. 205–216.
- Karttunen, L. 2007. Word play. *Computational Linguistics* 33 (4). 443–467.
- Katamba, F.X. (ed). 2003. *Morphology: Critical Concepts in Linguistics, VI: Morphology: Its Place in the Wider Context*. London, U.K.: Routledge.
- Kay, M. 2004. Introduction to Mitkov (ed.). xvii–xx.
- Kiraz, G.A. 2000. Multitiered nonlinear morphology using multitape finite automata. *Computational Linguistics* 26 (1). 77–105.
- Kiraz, G. 2008. Book review of *Arabic Computational Morphology: Knowledge-Based and Empirical Methods*. *Computational Linguistics* 34 (3). 459–462.
- Koskenniemi, K. 1983. *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production*. Publication 11, Department of General Linguistics, Helsinki, Finland: University of Helsinki.
- Krieger, H.U. and Nerbonne, J. 1992. Feature-based inheritance networks for computational lexicons. In: Briscoe, E., de Paiva, V., and Copestake, A. (eds). *Inheritance, Defaults and the Lexicon*. Cambridge, U.K.: Cambridge University Press. pp. 90–136.
- Kilbury, J. 2001. German noun inflection revisited. *Journal of Linguistics* 37 (2). 339–353.
- Langer, H. 1994. Reverse queries in DATR. *COLING-94*. Morristown, NJ: ACL. pp. 1089–1095.
- Mel'čuk, I. 2006. *Aspects of the Theory of Morphology*. Trends in Linguistics 146. Berlin/New York: Mouton de Gruyter.
- Mitkov, R. (ed). 2004. *The Oxford Handbook of Computational Linguistics*. Oxford, U.K.: Oxford University Press.
- Moreno-Sandoval, A. and Goñi-Menoyo, J.M. 2002. Spanish inflectional morphology in DATR. *Journal of Logic, Language and Information*. 11 (1). 79–105.
- Oncina, J., Garcia, P., and Vidal, P. 1993. Learning subsequential transducers for pattern recognition tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15. 448–458.
- Pollard, C. and Sag, I.A. 1994. *Head-Driven Phrase Structure Grammar*. Chicago, IL: University of Chicago Press.
- Pretorius, L. and Bosch, S.E. 2003. Finite-state morphology: An analyzer for Zulu. *Machine Translation* 18 (3). 195–216.
- Roark, B. and Sproat, R. 2007. *Computational Approaches to Syntax*. Oxford, U.K.: Oxford University Press.
- Sag, I., Wasow, T., and Bender, E.M. (eds). 2003. *Syntactic Theory: A Formal Introduction*. Stanford, CA: CSLI.
- Soudi, A. et al. (eds). 2007. *Arabic Computational Morphology: Knowledge-Based and Empirical Methods*. Dordrecht: Springer.

- Sproat, R. 1997. *Multilingual Text to Speech Synthesis: The Bell Labs Approach*. Dordrecht, the Netherlands: Kluwer.
- Sproat, R. 2000. Lexical analysis. In Dale, R. et al. (eds). pp. 37–58.
- Sproat, R. 2007. Preface. In: Soudi, A. et al (eds). *Arabic Computational Morphology: Knowledge-Based and Empirical Methods*. Dordrecht, the Netherlands: Springer. pp. vii–viii.
- Stewart, T. and Stump, G. 2007. Paradigm function morphology and the morphology-syntax interface. In: Ramchand, G. and Reiss, C. (eds). *Linguistic Interfaces*. Oxford, U.K.: Oxford University Press. pp. 383–421.
- Stump, G. 2001. *Inflectional Morphology*. Cambridge, U.K.: Cambridge University Press.
- Stump, G. 1993. On rules of referral. *Language* 69. 449–79.

Stump, G. 2006. Heterocclisis and paradigm linkage. *Language* 82, 279–322.

Timberlake, A.A. 2004. *A Reference Grammar of Russian*. Cambridge, U.K.: Cambridge University Press.

Tzoukermann, E., Klavans, J., and Strzalkowski, T. 2003. Information retrieval. In: Mitkov (ed.) 529–544.

UiDhonnchadha, E., NicPhaidin, C., and van Genabith, J. 2003. Design, implementation and evaluation of an inflectional morphology finite-state transducer for Irish. *Machine Translation* 18 (3). 173–193.

Yarowski, D. and Wicentowski, R. 2001. Minimally supervised morphological analysis by multimodal alignment. *Proceedings of the 38th ACL*. Morristown, NJ: ACL. pp. 207–216.

Yona, S. and Wintner, S. 2007. A finite-state morphological grammar of Hebrew. *Natural Language Engineering* 14. 173–190.

4

Syntactic Parsing

Introduction 59

4.2 Background 61

Context-Free Grammars · Example Grammar · Syntax Trees · Other Grammar Formalisms · Basic Concepts in Parsing

4.3 The Cocke–Kasami–Younger Algorithm 65

Handling Unary Rules · Example Session · Handling Long Right-Hand Sides

4.4 Parsing as Deduction 67

Deduction Systems · The CKY Algorithm · Chart Parsing · Bottom-Up Left-Corner Parsing · Top-Down Earley-Style Parsing · Example Session · Dynamic Filtering

4.5 Implementing Deductive Parsing 72

Agenda-Driven Chart Parsing · Storing and Retrieving Parse Results

4.6 LR Parsing. 73

The LR(0) Table · Deterministic LR Parsing · Generalized LR Parsing · Optimized GLR Parsing

4.7 Constraint-Based Grammars 77

Overview · Unification · Tabular Parsing with Unification

4.8 Issues in Parsing 79

Robustness · Disambiguation · Efficiency

4.9 Historical Notes and Outlook 83

Acknowledgments 84

References 84

Peter Ljunglöf
University of Gothenburg

Mats Wirén
Stockholm University

4.1

4.1 Introduction

This chapter presents basic techniques for grammar-driven natural language parsing, that is, analyzing a string of words (typically a sentence) to determine its structural description according to a formal grammar. In most circumstances, this is not a goal in itself but rather an intermediary step for the purpose of further processing, such as the assignment of a meaning to the sentence. To this end, the desired output of grammar-driven parsing is typically a hierarchical, syntactic structure suitable for semantic interpretation (the topic of Chapter 5). The string of words constituting the input will usually have been processed in separate phases of tokenization (Chapter 2) and lexical analysis (Chapter 3), which is hence not part of parsing proper.

To get a grasp of the fundamental problems discussed here, it is instructive to consider the ways in which parsers for natural languages differ from parsers for computer languages (for a related discussion, see Steedman 1983, Karttunen and Zwicky 1985). One such difference concerns the power of the grammar formalisms used—the *generative capacity*. Computer languages are usually designed so as to permit encoding by unambiguous grammars and parsing in linear time of the length of the input. To this end,

carefully restricted subclasses of context-free grammar (CFG) are used, with the syntactic specification of ALGOL 60 (Backus et al. 1963) as a historical exemplar. In contrast, natural languages are typically taken to require more powerful devices, as first argued by Chomsky (1956).^{*} One of the strongest cases for expressive power has been the occurrence of long-distance dependencies, as in English *wh*-questions:

Who did you sell the car to_ ? (4.1)

Who do you think that you sold the car to_ ? (4.2)

Who do you think that he suspects that you sold the car to_ ? (4.3)

In (4.1) through (4.3) it is held that the noun phrase “who” is displaced from its canonical position (indicated by “_”) as indirect object of “sell.” Since there is no clear limit as to how much material may be embedded between the two ends, as suggested by (4.2) and (4.3), linguists generally take the position that these dependencies might hold at unbounded distance. Although phenomena like this have at times provided motivation to move far beyond context-free power, several formalisms have also been developed with the intent of making minimal increases to expressive power (see Section 4.2.4). A key reason for this is to try to retain efficient parsability, that is, parsing in polynomial time of the length of the input. Additionally, for the purpose of determining the expressive power needed for linguistic formalisms, *strong generative capacity* (the structural descriptions assigned by the grammar) is usually considered more relevant than *weak generative capacity* (the sets of strings generated); compare Chomsky (1965, pp. 60–61) and Joshi (1997).

A second difference concerns the extreme structural *ambiguity* of natural language. At any point in a pass through a sentence, there will typically be several grammar rules that might apply. A classic example is the following:

Put the block in the box on the table (4.4)

Assuming that “put” subcategorizes for two objects, there are two possible analyses of (4.4):

Put the block [in the box on the table] (4.5)

Put [the block in the box] on the table (4.6)

If we add another prepositional phrase (“in the kitchen”), we get five analyses; if we add yet another, we get 14, and so on. Other examples of the same phenomenon are conjuncts and nominal compounding. As discussed in detail by Church and Patil (1982), “every-way ambiguous” constructions of this kind have a number of analyses that grows exponentially with the number of added components. Even though only one of them may be appropriate in a given context, the purpose of a general grammar might be to capture what is possible in *any* context. As a result of this, even the process of just returning all the possible analyses would lead to a combinatorial explosion. Thus, much of the work on parsing — hence, much of the following exposition — deals somehow or the other with ways in which the potentially enormous search spaces can be efficiently handled, and how the most appropriate analysis can be selected (*disambiguation*). The latter problem also leads naturally to extensions of grammar-driven parsing with statistical inference, as dealt with in Chapter 11.

A third difference stems from the fact that natural language data are inherently *noisy*, both because of errors (under some conception of “error”) and because of the ever persisting incompleteness of lexicon and grammar relative to the unlimited number of possible utterances which constitute the language. In contrast, a computer language has a complete syntax specification, which means that by definition all correct input strings are parsable. In natural language parsing, it is notoriously difficult to distinguish

* For a background on formal grammars and formal-language theory, see Hopcroft et al. (2006).

whether a failure to produce a parsing result is due to an error in the input or to the lack of coverage of the grammar, also because a natural language by its nature has no precise delimitation. Thus, input not licensed by the grammar may well be perfectly adequate according to native speakers of the language. Moreover, input containing errors may still carry useful bits of information that might be desirable to try to recover. *Robustness* refers to the ability of always producing *some* result in response to such input (Menzel 1995).

The rest of this chapter is organized as follows. Section 4.2 gives a background on grammar formalisms and basic concepts in natural language parsing, and also introduces a small CFG that is used in examples throughout. Section 4.3 presents a basic tabular algorithm for parsing with CFG, the Cocke–Kasami–Younger algorithm. Section 4.4 then describes the main approaches to tabular parsing in an abstract way, in the form of “parsing as deduction,” again using CFG. Section 4.5 discusses some implementational issues in relation to this abstract framework. Section 4.6 then goes on to describing LR parsing, and its nondeterministic generalization GLR parsing. Section 4.7 introduces a simple form of constraint-based grammar and describes tabular parsing using this kind of grammar formalism. Section 4.8 discusses in some further depth the three main challenges in natural language parsing that have been touched upon in this introductory section – robustness, disambiguation, and efficiency. Finally, Section 4.9 provides some brief historical notes on parsing relative to where we stand today.

4.2 Background

This section introduces grammar formalisms, primarily CFGs, and basic parsing concepts, which will be used in the rest of this chapter.

4.2.1 Context-Free Grammars

Ever since its introduction by Chomsky (1956), CFG has been the most influential grammar formalism for describing language syntax. This is not because CFG has been generally adopted as such for linguistic description, but rather because most grammar formalisms are derived from or can somehow be related to CFG. For this reason, CFG is often used as a base formalism when parsing algorithms are described.

The standard way of defining a CFG is as a tuple $G = (\Sigma, N, S, R)$, where Σ and N are disjoint finite sets of *terminal* and *nonterminal* symbols, respectively, and $S \in N$ is the *start symbol*. The nonterminals are also called *categories*, and the set $V = N \cup \Sigma$ contains the symbols of the grammar. R is a finite set of *production rules* of the form $A \rightarrow \alpha$, where $A \in N$ is a nonterminal and $\alpha \in V^*$ is a sequence of symbols.

We use capital letters $A, B, C, \dots$ for nonterminals, lower-case letters $s, t, w, \dots$ for terminal symbols, and uppercase $X, Y, Z, \dots$ for general symbols (elements in V). Greek letters $\alpha, \beta, \gamma, \dots$ will be used for sequences of symbols, and we write ϵ for the empty sequence.

The rewriting relation $\Rightarrow$ is defined by $\alpha B \gamma \Rightarrow \alpha \beta \gamma$ if and only if $B \rightarrow \beta$. A *phrase* is a sequence of terminals $\beta \in \Sigma^*$ such that $A \Rightarrow \dots \Rightarrow \beta$ for some $A \in N$. Accordingly, the term *phrase-structure grammar* is sometimes used for grammars with at least context-free power. The sequence of rule expansions is called a *derivation* of β from A . A (*grammatical*) *sentence* is a phrase that can be derived from the start symbol S . The *string language* $L(G)$ accepted by G is the set of sentences of G .

Some algorithms only work for particular *normal forms* of CFGs:

- In Section 4.3 we will use grammars in *Chomsky normal form* (CNF). A grammar is in CNF when

each rule is either (i) a unary terminal rule of the form $A \rightarrow w$, or (ii) a binary nonterminal rule of the form $A \rightarrow BC$. It is always possible to transform a grammar into CNF such that it accepts

S	→	NP	VP	Det	→	a an the
NP	→	Det	NBar	Adj	→	old
NBar	→	Adj	Noun	Noun	→	man men ship ships
NBar	→	Noun		Verb	→	man mans
NBar	→	Adj				
VP	→	Verb				
VP	→	Verb	NP			

FIGURE 4.1 Example grammar.

thesame language.* However, the transformation can change the structure of the grammar quite radically; e.g., if the original grammar has n rules, the transformed version may in the worst case have $O(n^2)$ rules (Hopcroft et al. 2006).

- We can relax this normal form by allowing (iii) unary nonterminal rules of the form $A \rightarrow B$. The transformation to this form is much simpler, and the transformed grammar is structurally closer; e.g., the transformed grammar will have only $O(n)$ rules. This relaxed variant of CNF is also used in Section 4.3.
- In Section 4.4 we relax the normal form even further, such that each rule is either (i) a unary terminal rule of the form $A \rightarrow w$, or (ii) a nonempty nonterminal rule of the form $A \rightarrow B_1 \cdots B_d$ ($d > 0$).
- In Section 4.6, the only restriction is that the rules are nonempty.

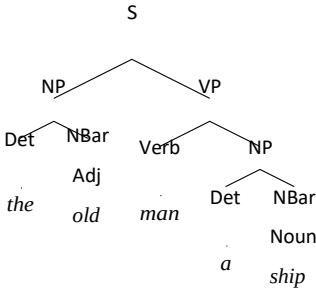
We will not describe how transformations are carried out here, but refer to any standard textbook on formal languages, such as Hopcroft et al. (2006).

4.2.2 Example Grammar

Throughout this chapter we will make use of a single (toy) grammar in our running examples. The grammar is shown in Figure 4.1, and is on CNF relaxed according to the first relaxation condition above. Thus it only contains unary and binary nonterminal rules, and unary terminal rules. The right-hand sides of the terminal rules correspond to *lexical items*, whereas the left-hand sides are *preterminal* (or *part-of-speech*) symbols. In practice, lexical analysis is often carried out in a phase distinct from parsing (as described in Chapter 3); the preterminals then take the role of terminals during parsing. The example grammar is lexically ambiguous, since the word “man” can be a noun as well as a verb. Hence, the *garden path* sentence “the old man a ship,” as well as the more intuitive “the old men man a ship,” can be recognized using this grammar.

4.2.3 Syntax Trees

The standard way to represent the syntactic structure of a grammatical sentence is as a *syntax tree*, or a *parse tree*, which is a representation of all the steps in the derivation of the sentence from the root node. This means that each internal node in the tree represents an application of a grammar rule. The syntax tree of the example sentence “the old man a



ship” is shown in Figure 4.2. Note that the tree is drawn upside-down, with the root of the tree at the top and

the leaves at the bottom.

FIGURE 4.2 Syntax tree of the sentence
“the old man a ship.”

* Formally, only grammars that do not accept the empty string can be transformed into CNF, but from a practical point of view we can disregard this, as we are not interested in empty string languages.

Another representation, which is commonly used in running text, is as a *bracketed* sentence, where the brackets are labeled with nonterminals:

[_S [_{NP} [_{Det} *the*] [_{NBar} [_{Adj} *old*]]] [_{VP} [_{Verb} *man*] [_{NP} [_{Det} *a*] [_{NBar} [_{Noun} *ship*]]]]]

4.2.4 Other Grammar Formalisms

In practice, pure CFG is not widely used for developing natural language grammars (though grammar-based language modeling in speech recognition is one such case; see Chapter 15). One reason for this is that CFG is not expressive enough—it cannot describe all peculiarities of natural language, e.g., Swiss-German or Dutch scrambling (Shieber 1985a), or Scandinavian long-distance dependencies (Kaplan and Zaenen 1995). But the main practical reason is that it is difficult to use; e.g., agreement, inflection, and other common phenomena are complicated to describe using CFG.

The example grammar in Figure 4.1 is overgenerating—it recognizes both the noun phrases “a men” and “an man,” as well as the sentence “the men mans a ship.” However, to make the grammar syntactically correct, we must duplicate the categories Noun, Det, and NP into singular and plural versions. All grammar rules involving these categories must be duplicated too. And if the language is, e.g., German, then Det and Noun have to be inflected on number (SING/PLUR), gender (FEM/NEUTR/MASC) and, case (NOM/ACC/DAT/GEN).

Ever since the late 1970s, a number of extensions to CFGs have emerged, with different properties. Some of these formalisms, for example, Regulus and Generalized Phrase-Structure Grammar (GPSG), are context-free-equivalent, meaning that grammars can be compiled to an equivalent CFG which then can be used for parsing. Other formalisms, such as Head-driven Phrase-Structure Grammar (HPSG) and Lexical-Functional Grammar (LFG), have more expressive power, but their similarities with CFG can still be exploited when designing tailor-made parsing algorithms.

There are also several grammar formalisms (e.g., categorial grammar, TAG, dependency grammar) that have not been designed as extensions of CFG, but have other pedigrees. However, most of them have been shown later to be equivalent to CFG or some CFG extension. This equivalence can then be exploited when designing parsing algorithms for these formalisms.

Mildly Context-Sensitive Grammars

According to Chomsky’s hierarchy of grammar formalisms (Chomsky 1959), the next major step after context-free grammar is *context-sensitive grammar*. Unfortunately, this step is substantial; arguably, context-sensitive grammars can express an unnecessarily large class of languages, with the drawback that parsing is no longer polynomial in the length of the input. Joshi (1985) suggested the notion of *mild context-sensitivity* to capture the precise formal power needed for defining natural languages. Roughly, a grammar formalism is regarded as mildly context-sensitive (MCS) if it can express some linguistically motivated non-context-free constructs (multiple agreement, crossed agreement, and duplication), and can be parsed in polynomial time with respect to the length of the input.

Among the most restricted MCS formalisms are Tree-Adjoining Grammar (TAG; Joshi et al. 1975, Joshi and Schabes 1997) and Combinatory Categorical Grammar (CCG; Steedman 1985, 1986), which are equivalent to each other (Vijay-Shanker and Weir 1994). Extending these formalisms we obtain a hierarchy of MCS grammar formalisms, with an upper bound in the form of Linear Context-Free Rewriting Systems (LCFRS; Vijay-Shanker et al. 1987), Multiple Context-Free Grammar (MCFG; Seki et al. 1991), and Range Concatenation Grammar (RCG; Boullier 2004), among others.

Constraint-Based Formalisms

A key characteristic of constraint-based grammars is the use of feature terms (sets of attribute–value pairs)

for the description of linguistic units, rather than atomic categories as in CFG. Feature terms are partial (underspecified) in the sense that new information may be added as long as it is compatible with old

information. *Regulus* (Rayner et al. 2006) and *GPSG* (Gazdar et al. 1985) are examples of constraint-based formalisms that are context-free-equivalent, whereas *HPSG* (Pollard and Sag 1994) and *LFG* (Bresnan 2001) are strict extensions of CFG. Not only CFG can be augmented with feature terms—constraint-based variants of, e.g., TAG and dependency grammars also exist. Constraint-based grammars are further discussed in Section 4.7.

Immediate Dominance/Linear Precedence

When describing languages with a relatively free word order, such as Latin, Finnish, or Russian, it can be fruitful to separate *immediate dominance* (ID; the parent–child relation) from *linear precedence* (LP; the linear order between the children) within phrases. The first formalism to make use of the ID/LP distinction was *GPSG* (Gazdar et al. 1985), and it has also been used in *HPSG* and other recent grammar formalisms. The main problem with ID/LP formalisms is that parsing can become very expensive. Some work has therefore been done to devise ID/LP formalizations that are easier to parse (Nederhof and Satta 2004a; Daniels and Meurers 2004).

Head Grammars

Some linguistic theories make use of the notion of the syntactic *head* of a phrase; e.g., the head of a verb phrase could be argued to be the main verb, whereas the head of a noun phrase could be the main noun. The simplest head grammar formalism is obtained by marking one right-hand side symbol in each context-free rule; more advanced formalisms include *HPSG*. The head information can, e.g., be used for driving the parser by trying to find the head first and then its arguments (Kay 1989).

Lexicalized Grammars

The nonterminals in a CFG do not depend on the lexical words at the surface level. This is a standard problem for *PP attachment*—which noun phrase or verb phrase constituent a specific prepositional phrase should be attached to. For example, considering a sentence beginning with “I bought a book . . .,” it is clear that a following PP “... with my credit card” should be attached to the verb “bought,” whereas the PP “... with an interesting title” should attach to the noun “book.” To be able to express such lexical syntactic preferences, CFGs and other formalisms can be lexicalized in different ways (Joshi and Schabes 1997, Eisner and Satta 1999, Eisner 2000).

Dependency Grammars

In contrast to constituent-based formalisms, *dependency grammar* lacks phrasal nodes; instead the structure consists of lexical elements linked by binary dependency relations (Tesnière 1959, Nivre 2006). A dependency structure is a directed acyclic graph between the words in the surface sentence, where the edges are labeled with syntactic functions (such as SUBJ, OBJ, MOD, etc.). Apart from this basic idea, the dependency grammar tradition constitutes a diverse family of different formalisms that can impose different constraints on the dependency relation (such as allowing or disallowing crossing edges), and incorporate different extensions (such as feature terms).

Type-Theoretical Grammars

Some formalisms are based on dependent type theory utilizing the Curry–Howard isomorphism between propositions and types. These formalisms include ALE (Carpenter 1992), Grammatical Framework (Ranta 1994, 2004, Ljunglöf 2004), and Abstract Categorical Grammar (de Groote 2001).

4.2.5 Basic Concepts in Parsing

A *recognizer* is a procedure that determines whether or not an input sentence is grammatical according to

the grammar (including the lexicon). A *parser* is a recognizer that produces associated structural analyses

according to the grammar (in our case, parse trees or feature terms). A *robust parser* attempts to produce useful output, such as a partial analysis, even if the input is not covered by the grammar (see Section 4.8.1).

We can think of a grammar as inducing a search space consisting of a set of states representing stages of successive grammar-rule rewritings and a set of transitions between these states. When analyzing a sentence, the parser (recognizer) must rewrite the grammar rules in some sequence. A sequence that connects the state S , the string consisting of just the start category of the grammar, and a state consisting of exactly the string of input words, is called a *derivation*. Each state in the sequence then consists of a string over V^* and is called a *sentential form*. If such a sequence exists, the sentence is said to be grammatical according to the grammar.

Parsers can be classified along several dimensions according to the ways in which they carry out derivations. One such dimension concerns rule invocation: In a *top-down* derivation, each sentential form is produced from its predecessor by replacing one nonterminal symbol A by a string of terminal or nonterminal symbols $X_1 \cdots X_d$, where $A \rightarrow X_1 \cdots X_d$ is a grammar rule. Conversely, in a *bottom-up* derivation, each sentential form is produced by replacing $X_1 \cdots X_d$ with A given the same grammar rule, thus successively applying rules in the reverse direction.

Another dimension concerns the way in which the parser deals with ambiguity, in particular, whether the process is *deterministic* or *nondeterministic*. In the former case, only a single, irrevocable choice may be made when the parser is faced with local ambiguity. This choice is typically based on some form of lookahead or systematic preference.

A third dimension concerns whether parsing proceeds from left to right (strictly speaking front to back) through the input or in some other order, for example, inside-out from the right-hand-side heads.

4.3 The Cocke–Kasami–Younger Algorithm

The Cocke–Kasami–Younger (CKY, sometimes written CYK) algorithm, first described in the 1960s (Kasami 1965, Younger 1967), is one of the simplest context-free parsing algorithms. A reason for its simplicity is that it only works for grammars in CNF.

The CKY algorithm builds an upper triangular matrix T , where each cell $T_{i,j}$ ($0 \leq i < j \leq n$) is a set of nonterminals. The meaning of the statement $A \in T_{i,j}$ is that A spans the input words $w_{i+1} \cdots w_j$, or written more formally, $A \Rightarrow^* w_{i+1} \cdots w_j$.

CKY is a purely bottom-up algorithm consisting of two parts. First we build the lexical cells $T_{i-1,i}$ for the input word w_i by applying the lexical grammar rules, then the nonlexical cells $T_{i,k}$ ($i < k - 1$) are filled by applying the binary grammar rules:

$$T_{i-1,i} = \{ A \mid A \rightarrow w_i \}$$

$$T_{i,k} = \bigcup \{ A \mid A \rightarrow BC, i < j < k, B \in T_{i,j}, C \in T_{j,k} \}$$

The sentence is recognized by the algorithm if $S \in T_{0,n}$, where S is the start symbol of the grammar.

To make the algorithm less abstract, we note that all cells $T_{i,j}$ and $T_{j,k}$ ($i < j < k$) must already be known when building the cell $T_{i,k}$. This means that we have to be careful when designing the i and k loops, so that smaller spans are calculated before larger spans.

One solution is to start by looping over the end node k , and then loop over the start node i in the *reverse* direction. The pseudo-code is as follows:

```

procedure CKY( $T, w_1 \cdots w_n$ )  $T_{i,j}$ 
   $:= \emptyset$  forall  $0 \leq i, j \leq n$  for  $i$ 
   $:= 1$  to  $n$  do

```

for all lexical rules $A \rightarrow w$ **do**

if $w = w_i$ **then** add A to $T_{i-1,i}$

```

for  $k := 2$  to  $n$  do
  for  $i := k - 2$  downto  $0$  do
    for  $j := i + 1$  to  $k - 1$  do
      for all binary rules  $A \rightarrow BC$  do
        if  $B \in T_{i,j}$  and  $C \in T_{j,k}$  then add  $A$  to  $T_{i,k}$ 

```

But there are also several alternative possibilities for how to encode the loops in the CKY algorithm; e.g., instead of letting the outer k loop range over end positions, we could equally well let it range over span lengths. We have to keep in mind, however, that smaller spans must be calculated before larger spans.

As already mentioned, the CKY algorithm can only handle grammars in CNF. Furthermore, converting a grammar to CNF is a bit complicated, and can make the resulting grammar much larger, as mentioned in Section 4.2.1. Instead we will show how to modify the CKY algorithm directly to handle unary grammar rules and longer right-hand sides.

4.3.1 Handling Unary Rules

The CKY algorithm can only handle grammars with rules of the form $A \rightarrow w$ and $A \rightarrow BC$. Unfortunately most practical grammars also contain lots of unary rules of the form $A \rightarrow B$. There are two possible ways to solve this problem. Either we transform the grammar into CNF, or we modify the CKY algorithm.

If $B \in T_{i,k}$ and there is a unary rule $A \rightarrow B$, then we should also add A to $T_{i,k}$. Furthermore, the unary rules can be applied after the binary rules, since binary rules only apply to smaller phrases. Unfortunately, we cannot simply loop over each unary rule $A \rightarrow B$ to test if $B \in T_{i,k}$. The reason for this is that we cannot possibly know in which order the unary rules will be applied, which means that we cannot know in which order we have to select the unary rules $A \rightarrow B$. Instead we need to add the *reflexive, transitive closure* $\text{UNARY-CLOSURE}(B) = \{A \mid A \Rightarrow^* B\}$ for each $B \in T_{i,k}$. Since there are only a finite number of nonterminals, $\text{UNARY-CLOSURE}()$ can be precompiled from the grammar into an efficient lookup table.

Now, the only thing we have to do is to map $\text{UNARY-CLOSURE}()$ onto $T_{i,k}$ within the k and i loops, and after the j loop (as well as onto $T_{i-1,j}$ after the lexical rules have been applied). The final pseudo-code for the extended CKY algorithm is as follows:

```

procedure  $\text{UNARY-CKY}(T, w_1 \dots w_n)$ 
   $T_{i,j} := \emptyset$  for all  $0 \leq i, j \leq n$ 
  for  $i := 1$  to  $n$  do
    for all lexical rules  $A \rightarrow w$  do
      if  $w = w_i$  then add  $A$  to  $T_{i-1,i}$ 
    for all  $B \in T_{i-1,i}$  do
      add  $\text{UNARY-CLOSURE}(B)$  to  $T_{i-1,i}$ 
  for  $k := 2$  to  $n$  do
    for  $i := k - 2$  downto  $0$  do
      for  $j := i + 1$  to  $k - 1$  do
        for all binary rules  $A \rightarrow BC$  do

```

```

    if  $B \in T_{i,j}$  and  $C \in T_{j,k}$  then add  $A$  to  $T_{i,k}$ 

for all  $B \in T_{i,k}$  do
    add  $\text{UNARY-CLOSURE}(B)$  to  $T_{i,k}$ 

```

4.3.2 Example Session

The final CKY matrix after parsing the example sentence “the old man a ship” is shown in Figure 4.3. In the initial lexical pass, the cells in the first diagonal are filled. For example, the cell $T_{2,3}$ is initialized to {Noun,Verb}, after which $\text{UNARY-CLOSURE}()$ adds NBar and VP to it.

	1	2	3	4	5
0	Det <i>the</i>	NP	NP, S		S
1		Adj, NBar <i>old</i>	NBar		
2			Noun, Verb, NBar, VP <i>man</i>	Det	NP
3				<i>a</i>	Noun, NBar
4					

ship

FIGURE 4.3 CKY matrix after parsing the sentence “the old man a ship.”

Then other cells are filled from left to right, bottom up. For example, when filling the cell $T_{0,3}$, we have already filled $T_{0,2}$ and $T_{1,3}$. Now, since $\text{Det} \in T_{0,1}$ and $\text{NBar} \in T_{1,3}$, and there is a rule $\text{NP} \rightarrow \text{Det NBar}$, NP is added to $T_{0,3}$. And since $\text{NP} \in T_{0,2}$, $\text{VP} \in T_{2,3}$, and $\text{S} \rightarrow \text{NP VP}$, the algorithm adds S to $T_{0,3}$.

4.3.3 Handling Long Right-Hand Sides

To handle longer right-hand sides (RHS), there are several possibilities. A straightforward solution is to add a new inner loop for each RHS length. This means that, e.g., ternary rules will be handled by the following loop inside the k , i , and j nested loops:

```

for  $k, i, j := \dots$  do
  for all binary rules  $\dots$  do  $\dots$ 
  for  $j^j := j + 1$  to  $k - 1$  do
    for all ternary rules  $A \rightarrow BCD$  do
      if  $B \in T_{i,j}$  and  $C \in T_{j,j^j}$  and  $D \in T_{j^j,k}$  then
        add  $A$  to  $T_{i,k}$ 

```

To handle even longer rules we need to add new inner loops inside the j^j loop. And for each nested loop, the parsing time increases. In fact, the worst case complexity is $O(n^{d+1})$, where d is the length of the longest right-hand side. This is discussed further in Section 4.8.3.

A more general solution is to replace each long rule $A \rightarrow B_1 \cdots B_d$ ($d > 2$) by the $d - 1$ binary rules $A \rightarrow B_1 X_2$, $X_2 \rightarrow B_2 X_3$, $\dots$, $X_{d-1} \rightarrow B_{d-1} B_d$, where each $X_i = (B_i \cdots B_n)$ is a new nonterminal. After this transformation the grammar only contains unary and binary rules, which can be handled by the extended CKY algorithm.

Another variant of the binary transform is to do the RHS transformations implicitly during parsing. This gives rise to the well-known chart parsing algorithms that we introduce in Section 4.4.

4.4 Parsing as Deduction

In this section we will use a general framework for describing parsing algorithms in a high-level manner. The framework is called *deductive parsing*, and was introduced by Pereira and Warren (1983); a related framework introduced later was the *parsing schemata* of Sikkel (1998). Parsing in this sense can be seen as “a deductive process in which rules of inference are used to derive statements about the grammatical status of strings from other such statements” (Shieber et al. 1995).

4.4.1 Deduction Systems

The statements in a deduction system are called *items*, and are represented by formulae in some formal language. The inference rules and axioms are written in natural deduction style and can have side conditions mentioning, e.g., grammar rules. The inference rules and axioms are rule schemata; in other words, they contain metavariables to be instantiated by appropriate terms when the rule is invoked. The set of items built in the deductive process is sometimes called a *chart*.

The general form of an inference rule is

$$\frac{e_1 \dots e_n \quad \varphi}{e}$$

where $e, e_1, \dots, e_n$ are items and φ is a side condition. If there are no antecedents (i.e., $n = 0$), the rule is called an *axiom*. The meaning of an inference rule is that whenever we have derived the items $e_1, \dots, e_n$, and the condition φ holds, we can also derive the item e . The inference rules are applied until no more items can be added. It does not make any difference in which order the rules are applied — the final chart is the reflexive, transitive closure of the inference rules. However, one important constraint is that the system is terminating, which is the case if the number of possible items is finite.

4.4.2 The CKY Algorithm

As a first example, we describe the extended CKY algorithm from Section 4.3.1 as a deduction system. The items are of the form $[i, k : A]$, corresponding to a nonterminal symbol A spanning the input words $w_{i+1} \dots w_k$. This is equivalent to the statement $A \in T_{i,k}$ in Section 4.3. We need three inference rules, of which one is an axiom.

Combine

$$\frac{\begin{array}{c} \Sigma i, j : B \quad \Sigma j, k : C \\ A \rightarrow BC \end{array}}{[i, k : A]} \quad (4.7)$$

If there is a B spanning the input positions $i - j$, and a C spanning $j - k$, and there is a binary rule $A \rightarrow BC$, we know that A will span the input positions $i - k$.

Unary Closure

$$\frac{[i, k : B]}{[i, k : A]} \quad A \rightarrow B \quad (4.8)$$

If we have a B spanning $i - k$, and there is a rule $A \rightarrow B$, then we also know that there is an A spanning $i - k$.

Scan

$$\frac{}{[i - 1, i : A]} \quad A \rightarrow w_i \quad (4.9)$$

Finally we need an axiom adding an item for each matching lexical rule.

Note that we do not have to say anything about the order in which the inference rules should be applied, as was the case when we presented the CKY algorithm in Section 4.3.

4.4.3 Chart Parsing

The CKY algorithm uses a bottom-up parsing strategy, which means that it starts by recognizing the lexical nonterminals, i.e., the nonterminals that occur as left-hand sides in unary terminal rules. Then

the algorithm recognizes the parents of the lexical nonterminals, and so on until it reaches the starting symbol.

A disadvantage of CKY is that it only works on restricted grammars. General CFGs have to be converted, which is not a difficult problem, but can be awkward. The parse results: also have to be back-translated into the original form. Because of this, one often implements more general parsing strategies instead.

In the following we give examples of some well-known parsing algorithms for CFGs. First we give a very simple algorithm, and then two refinements; Kilbury's bottom-up algorithm (Leiss 1990), and Earley's top-down algorithm (Earley 1970). The algorithms are slightly modified for presentational purposes, but their essence is still the same.

Parse Items

Parse items are of the form $\langle i, j : A \rightarrow \alpha \cdot \beta \rangle$ where $A \rightarrow \alpha\beta$ is a context-free rule, and $0 \leq i \leq j \leq n$ are positions in the input string. The meaning is that α has been recognized spanning $i - j$; i.e., $\alpha \Rightarrow^* w_{i+1} w_j$. If β is empty, the item is called *passive*. Apart from the logical meaning, the item also states that it is searching for β to span the positions j and k (for some k). The goal of the parsing process is to deduce an item representing that the starting category is found spanning the whole input string; such an item can be written $[0, n : S \rightarrow \alpha \cdot]$ in our notation.

To simplify presentation, we will assume that all grammars are of the relaxed normal form presented in Section 4.2.1, where each rule is either lexical $A \rightarrow w$ or nonempty $A \rightarrow B_1 B_2$. To extend the algorithms to cope with general grammars constitutes no serious problem.

The Simplest Chart Parsing Algorithm

Our first context-free chart parsing algorithm consists of three inference rules. The first two, Combine and Scan, remain the same in all our chart parsing variants; while the third, Predict, is very simple and will be improved upon later. The algorithm is also presented by Sikkel and Nijholt (1997), who call it *bottom-up Earley parsing*.

Combine

$$\frac{\langle i, j : A \rightarrow \alpha \cdot B \beta \rangle \quad \langle j, k : B \rightarrow \gamma \cdot \rangle}{\langle i, k : A \rightarrow \alpha B \beta \rangle} \quad (4.10)$$

The basis for all chart parsing algorithms is *the fundamental rule*; saying that if there is an active item looking for a category B spanning $i - j$, and there is a passive item for B spanning $j - k$, then the dot in the active item can be moved forward, and the new item will span the positions $i - k$.

Scan

$$\frac{}{\langle i - 1, i : A \rightarrow w_i \cdot \rangle} A \rightarrow w_i \quad (4.11)$$

This is similar to the scanning axiom of the CKY algorithm.

Predict

$$\frac{}{\langle i, i : A \rightarrow \cdot \beta \rangle} A \rightarrow \beta \quad (4.12)$$

This axiom takes care of introducing active items; each rule in the grammar is added as an active

item spanning $i - i$ for any possible input position $0 \leq i \leq n$.

The main problem with this algorithm is that prediction is “blind”; active items are introduced for every rule in the grammar, at all possible input positions. Only very few of these items will be used in later

inferences, which means that prediction infers a lot of useless items. The solution is to make prediction an inference rule instead of an axiom, so that an item is only predicted if it is potentially useful for already existing items.

In the rest of this section we introduce two basic prediction strategies, *bottom-up* and *top-down*.

4.4.4 Bottom-Up Left-Corner Parsing

The basic idea with bottom-up parsing is that we predict a grammar rule only when its first symbol has already been found. Kilbury's variant of bottom-up parsing (Leiss 1990) moves the dot in the new item forward one step. Since the first symbol in the right-hand side is called the left corner, the algorithm is sometimes called *bottom-up left-corner parsing* (Sikkel 1998).

Bottom-Up Predict

$$\frac{[i, k : B \rightarrow \gamma \cdot]}{[i, k : A \rightarrow B \cdot \beta]} A \rightarrow B\beta \quad (4.13)$$

Bottom-up prediction is like Combine for the first symbol on the right-hand side in a rule. If we have found a B spanning $i - k$, and there is a rule $A \rightarrow B\beta$, we can draw the conclusion that $A \rightarrow B \cdot \beta$ will span $i - k$.

Note that this algorithm does not work for grammars with ϵ -rules; there is no way an empty rule can be predicted. There are two possible ways ϵ -rules can be handled: (1) either convert the grammar to an equivalent ϵ -free grammar; or (2) add extra inference rules to handle ϵ -rules.

4.4.5 Top-Down Earley-Style Parsing

Earley prediction (Earley 1970) works in a top-down fashion; meaning that we start by stating that we want to find an S starting in position 0, and then move downward in the presumptive syntactic structure until we reach the lexical tokens.

Top-Down Predict

$$\frac{[i, k : B \rightarrow \gamma \cdot A\alpha]}{[k, k : A \rightarrow \cdot \beta]} A \rightarrow \beta \quad (4.14)$$

If there is an item looking for an A beginning in position k , and there is a grammar rule for A , we can add that rule as an empty active item starting and ending in k .

Initial Predict

$$\frac{}{[0, 0 : S \rightarrow \cdot \beta]} S \rightarrow \beta \quad (4.15)$$

Top-down prediction needs an active item to be triggered, so we need some way of starting the inference process. This is done by adding an active item for each rule of the starting category S , starting and ending in 0.

4.4.6 Example Session

The final charts after bottom-up and top-down parsing of the example sentence “the old man a ship” are shown in Figures 4.4 and 4.5. This is a standard way of visualizing a chart, as a graph where the items are

drawn as edges between the input positions. In the figures, the dotted and grayed-out edges correspond

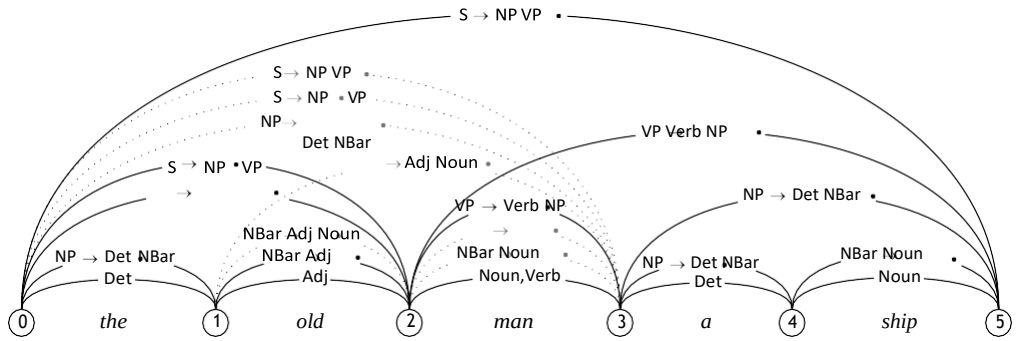


FIGURE 4.4 Final chart after bottom-up parsing of the sentence “the old man a ship.” The dotted edges are inferred but useless.

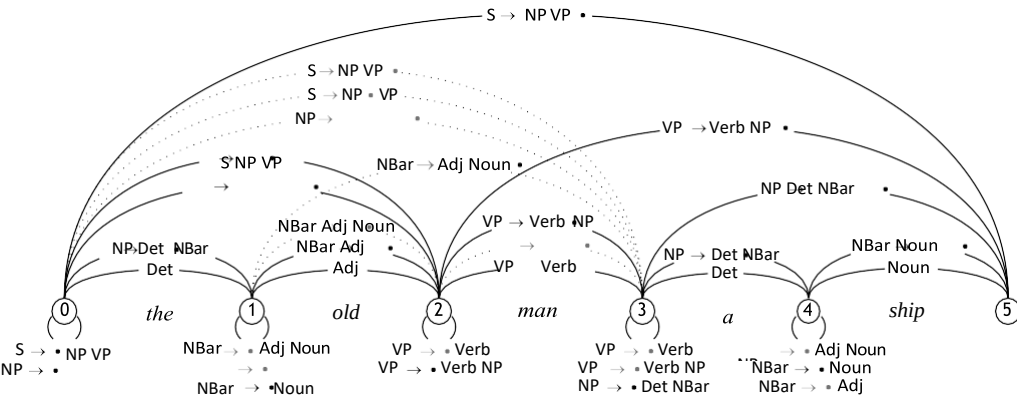


FIGURE 4.5 Final chart after top-down parsing of the sentence “the old man a ship.” The dotted edges are inferred but useless.

to useless items, i.e., items that are not used in any derivation of the final S item spanning the whole sentence.

The bottom-up chart contains the useless item $[2, 3 : \text{NBar} \rightarrow \text{Noun}]$, which the top-down chart does not contain. On the other hand, the top-down chart contains a lot of useless cyclic predictions. This suggests that both bottom-up and top-down parsing have their advantages and disadvantages, and that combining the strategies could be the way to go. This leads us directly into the next section about dynamic filtering.

4.4.7 Dynamic Filtering

Both the bottom-up and the top-down algorithms have disadvantages. Bottom-up prediction has no idea of what the final goal of parsing is, which means that it predicts items which will not be used in any derivation from the top node. Top-down prediction on the other hand never looks at the input words, which means that it predicts items that can never start with the next input word.

Note that these useless items do not make the algorithms incorrect in any way; they only decrease

parsing efficiency. There are several ways the basic algorithms can be optimized; the standard optimizations are by adding top-down and/or bottom-up *dynamic filtering* to the prediction rules.

Bottom-up filtering adds a side condition stating that a prediction is only allowed if the resulting item can start with the next input token. For this we make use of a function `FIRST()` that returns the set of terminals than can start a given symbol sequence. The only thing we have to do is to add a side condition $w_k \in \text{FIRST}(\beta)$ to top-down prediction (4.14), and bottom-up prediction (4.13), respectively. * Possible definitions of the function `FIRST()` can be found in standard textbooks (Aho et al. 2006, Hopcroft et al. 2006).

Top-down filtering In a similar way, we can add constraints for top-down filtering of the bottom-up strategy. This means that we only have to add a constraint to bottom-up prediction (4.13) that there is an item looking for a C , where $C \Rightarrow^* A\delta$ for some δ . This *left-corner* relation can be precompiled from the grammar, and the resulting parsing strategy is often called left-corner parsing (Sikkel and Nijholt 1997, Moore 2004).

Furthermore, both bottom-up and top-down filterings can be added as side-conditions to bottom-up prediction (4.13). Further optimizations in this direction, such as introducing special predict items and realizing the parser as an incremental algorithm, are discussed by Moore (2004).

4.5 Implementing Deductive Parsing

This section briefly discusses how to implement the deductive parsing framework, including how to store and retrieve parse results.

4.5.1 Agenda-Driven Chart Parsing

A deduction engine should infer all consequences of the inference rules. As mentioned above, the set of all resulting items is called a *chart*, and can be calculated using a forward-chaining deduction procedure. Whenever an item is added to the chart, its consequences are calculated and added. However, since one item can give rise to several new items, we need to keep track of the items that are waiting to be processed. New items are thus added to a separate agenda that is used for bookkeeping.

The idea is as follows: First we add all possible consequences of the axioms to the agenda. Then we remove one item e from the agenda, add it to the chart, and add all possible inferences that are triggered by e to the agenda. This second step is repeated until the agenda is empty.

Regarding efficiency, the bottleneck of the algorithm is searching the chart for items matching the inference rule. Because of this, the chart needs to be indexed for efficient antecedent lookup. Exactly what indexes are needed depend on the inference rules and will not be discussed here. For a thorough discussion about implementation issues, see Shieber et al. (1995).

4.5.2 Storing and Retrieving Parse Results

The set of syntactic analyses (or *parse trees*) for a given string is called a *parse forest*. The size of this set can be exponential in the length of the string, as mentioned in the introduction section. A classical example is a grammar for PP attachment containing the rules $\text{NP} \rightarrow \text{NPP}$ and $\text{PP} \rightarrow \text{Prep NP}$. In some pathological cases (i.e., when the grammar is cyclic), there might even be an infinite number of trees. The polynomial parse time complexity stems from the fact that the parse forest can be compactly stored in polynomial space.

A parse forest can be represented as a CFG recognizing the language consisting of only the input string (Bar-Hillel et al. 1964). The forest can then be further investigated to remove useless nodes, increase sharing, and reduce space complexity (Billot and Lang 1989).

* There is nothing that prevents us from adding a bottom-up filter to the combine rule (4.10) either. However, this filter is seldom used in practice.

Retrieving a single parse tree from a (suitably reduced) forest is efficient, but the problem is to decide which tree is the best one. We do not want to examine exponentially many trees, but instead we want a clever procedure for directly finding the best tree. This is the problem of disambiguation, which is discussed in Section 48.2 and in Chapter 11.

4.6 LR Parsing

Instead of using the grammar directly, we can precompile it into a form that makes parsing more efficient. One of the most common strategies is LR parsing, which was introduced by Knuth (1965). It is mostly used for deterministic parsing of formal languages such as programming languages, but was extended to nondeterministic languages by Lang (1974) and Tomita (1985, 1987).

One of the main ideas of LR parsing is to handle a number of grammar rules simultaneously by merging common subparts of their right-hand sides, rather than attempting one rule at a time. An LR parser compiles the grammar into a finite automaton, augmented with *reductions* for capturing the nesting of nonterminals in a syntactic structure, making it a kind of push-down automaton (PDA). The automaton is called an LR automaton, or an LR table.

4.6.1 The LR(0) Table

LR automata can be constructed in several different ways. The simplest construction is the LR(0) table, which uses no *lookahead* when it constructs its states. In practice, most LR algorithms use SLR(1) or LALR(1) tables, which utilize a lookahead of one input symbol. Details of how to construct these automata are, e.g., given by Aho et al. (2006). Our LR(0) construction is similar to the one by Nederhof and Satta (2004b).

States

The states in an LR table are sets of dotted rules $A \rightarrow \alpha \cdot \beta$. The meaning of being in a state is that any of the dotted rules in the state can be the correct one, but we have not decided yet.

To build an LR(0) table we do the following. First we have to define the function $\text{PREDICT-CLOSURE}(q)$, which is the smallest set such that:

- $q \subseteq \text{PREDICT-CLOSURE}(q)$, and
- if $(A \rightarrow \alpha \cdot B\beta) \in \text{PREDICT-CLOSURE}(q)$,
then $(B \rightarrow \cdot \gamma) \in \text{PREDICT-CLOSURE}(q)$ for all $B \rightarrow \gamma$

Transitions

Transitions between states are defined by the function GOTO , taking a grammar symbol as argument. The function is defined as

$$\text{GOTO}(q, X) = \text{PREDICT-CLOSURE}(\{A \rightarrow \alpha X \cdot \beta \mid A \rightarrow \alpha \cdot X\beta \in q\})$$

The idea is that all dotted rules $A \rightarrow \alpha \cdot X\beta$ will survive to the next state, with the dot moved forward one step. To this the closure of all top-down predictions are added. The initial state q_{init} of the LR table contains predictions of all S rules:

$$q_{\text{init}} = \text{PREDICT-CLOSURE}(\{S \rightarrow \cdot \gamma \mid S \rightarrow \gamma\})$$

We also need a special final state q_{final} that is reachable from the initial state by the dummy transition $\text{GOTO}(q_{\text{final}}, S)$. Figure 4.6 contains the resulting LR(0) table of the example grammar in Figure 4.1. The reducible states, marked with a thicker border in the figure, are the states that contain passive dotted rules,

i.e., rules of the form $A \rightarrow \alpha$. For simplicity we have not included the lexical rules in the LR table.

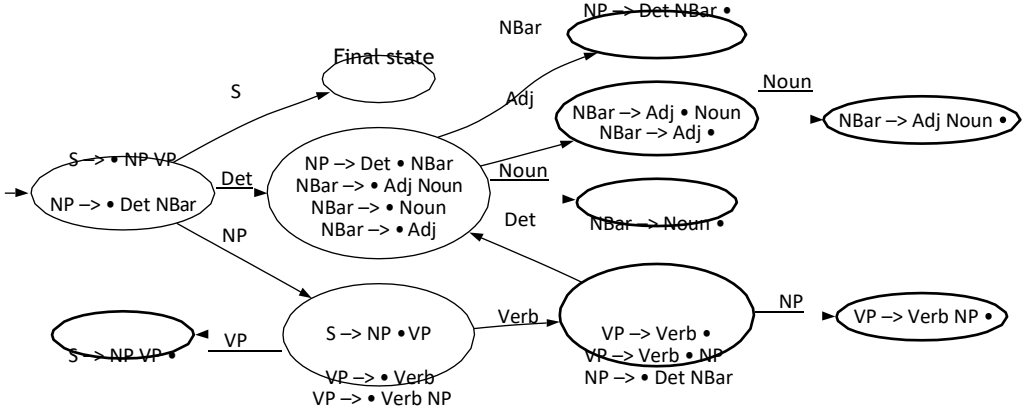


FIGURE 4.6 Example LR(0) table for the grammar in Figure 4.1.

4.6.2 Deterministic LR Parsing

An LR parser is a shift-reduce parser (Aho et al. 2006) that uses the transitions of the LR table to push states onto a stack. When the parser is in a reducible state, containing a rule $B \rightarrow \beta$, it pops $|\beta|$ states off the stack and shifts to a new state by the symbol B .

In our setting we do not use LR states directly, but instead LR states indexed by input position, which we write as $\sigma = q@i$. An LR stack $\omega = \sigma_1 \cdot \dots \cdot \sigma_n$ is a sequence of indexed LR states. There are three basic operations:

$$\text{TOP}(\omega) = \sigma \quad (\text{where } \omega = \omega^i \sigma)$$

$$\text{POP}(\omega) = \omega^i \quad (\text{where } \omega = \omega^i \sigma)$$

$$\text{PUSH}(\omega, \sigma) = \omega \sigma$$

The parser starts with a stack containing only the initial state in position 0. Then it shifts the next input symbol, and pushes the new state onto the stack. Note the difference with traversing a finite automaton; we do not forget the previous state, but instead push the new state on top. This way we know how to go backward in the automaton, which we cannot do in a finite automaton.

After shifting, we try to reduce the stack as often as possible, and then we shift the next input token, reduce and continue until the input is exhausted. The parsing has succeeded if we end up with q_{final} as the top state in the stack.

function $\text{LR}(w_1 \cdot \dots \cdot w_n)$

$\omega := (q_{\text{init}}@0)$

for $i := 1$ **to** n **do**

$$\omega := \text{REDUCE}(\text{SHIFT}(\omega, w_i @ i))$$

if $\text{TOP}(\omega) = q_{\text{final}} @ n$ **then** success **else** failure

Shifting a Symbol

To shift a symbol X onto the stack ω , we follow the edge labeled X from the current state $\text{TOP}(\omega)$, and push the new state onto the stack:

* We will abuse the function $\text{TOP}(\omega)$ by sometimes letting it denote the indexed state $q @ i$ and sometimes the LR state q . Furthermore we will write POP^n for n applications of POP .

```
function SHIFT( $\omega$ ,  $X@i$ )
```

```
     $\sigma_{\text{next}} := \text{GOTO}(\text{TOP}(\omega), X) @ i$ 
```

```
return PUSH( $\omega$ ,  $\sigma_{\text{next}}$ )
```

Reducing the Stack

When the top state of the stack contains a rule $A \rightarrow B_1 B_d \cdot$, the nonterminal A has been recognized. The only way to reach that state is from a state containing the rule $A \rightarrow B_1 B_{d-1} \cdot B_d$, which in turn is reached from a state containing $A \rightarrow B_1 B_{d-2} \cdot B_{d-1} B_d$, and so on d steps back in the stack. This state, d steps back, contains the predicted rule $A \rightarrow \cdot B_1 B_d$. But there is only one way this rule could have been added to the state—as a prediction from a rule $C \rightarrow \alpha \cdot A \beta$. So, if we remove d states from the stack (getting the popped stack ω_{red}), we reach a state that has an A transition.* And since we started this paragraph by knowing that A was just recognized, we can shift the popped stack ω_{red} . This whole sequence (popping the stack and then shifting) is called a *reduction*.

However, it is not guaranteed that we can stop here. It is possible that we, after shifting A onto the popped stack, enter a new reducible state, and we can do the whole thing again. This is done until there are no more possible reductions:

```
function REDUCE( $\omega$ )
```

```
     $q_{\text{top}}@i := \text{TOP}(\omega)$ 
```

```
    if  $(A \rightarrow B_1 \cdots B_d \cdot) \in q_{\text{top}}$  then
```

```
         $\omega_{\text{red}} := \text{POP}(\omega)$ 
```

```
        return REDUCE(SHIFT( $\omega_{\text{red}}$ ,  $A@i$ ))
```

```
    else
```

```
        return  $\omega$ 
```

Ungrammaticality and Ambiguities

The LR automaton can only handle grammatically correct input. If the input is ungrammatical, we might end up in a state where we can neither reduce nor shift. In this case we have to stop parsing and report an error.

The automaton could also contain nondeterministic choices, even on unambiguous grammars. Thus we might enter a state where it is possible to both shift and reduce at the same time (or reduce in two different ways). In deterministic LR parsing this is called a shift/reduce (or reduce/reduce) conflict, which is considered to be a problem in the grammar. However, since natural language grammars are inherently ambiguous, we have to change the algorithm to handle these cases.

4.6.3 Generalized LR Parsing

To handle nondeterminism, the top-level LR algorithm does not have to be changed much, and only small changes have to be made on the shift and reduce functions. Conceptually, we can think of a “stack” for nondeterministic LR parsing as a set K of ordinary stacks ω , which we reduce and shift in parallel. When reducing the stack set, we perform all possible reductions on the elements and take the union of

the results. This means that the number of stacks increases, but (hopefully) some of these stacks will be lost when shifting.

The top-level parsing function LR remains as before, with the slight modification that the initial stack set is the singleton set $\{(q_{\text{init}})\}$, and that the final stack set should contain some stack whose top state is $q_{\text{final}}@n$.

* Note that if A is the starting symbol S , and $\text{TOP}(\omega_{\text{red}})$ is the initial state $q_{\text{init}}@0$, then there will in fact *not* be any rule

$C \rightarrow \alpha \cdot S \beta$ in that state. But in this case there is a dummy transition to the final state q_{final} , so we can still shift over S .

Using a Set of Stacks

The basic stack operations `POP` and `PUSH` are straightforward to generalize to sets:

$$\text{POP}(K) = \{\omega \mid \omega\sigma \in K\}$$

$$\text{PUSH}(K, \sigma) = \{\omega\sigma \mid \omega \in K\}$$

However, there is one problem when trying to obtain the `TOP` state — since there are several stacks, there can be several different top states. And the `TOP` operation cannot simply return an unstructured set of states, since we have to know which stacks correspond to which top state. Our solution is to introduce an operation `TOP-PARTITION` that returns a *partition* of the set of stacks, where all stacks in each part have the same unique top state. The simplest definition is to just make each stack a part of its own, $\text{TOP-PARTITION}(K) = \{\{\omega\} \mid \omega \in K\}$, but there are several other possible definitions. Now we can define the `TOP` operation to simply return the top state of any stack in the set, since it will be unique.

$$\text{TOP}(K) = \sigma \quad (\text{where } \omega\sigma \in K)$$

Shifting

The difference compared to deterministic shift is that we loop over the stack partitions, and shift each partition in parallel, returning the union of the results:

function `SHIFT`($K, X@i$)

```

 $K^j := \emptyset$ 
for all  $\omega_{\text{part}} \in \text{TOP-PARTITION}(K)$  do  $\sigma_{\text{next}}$ 
  :      = GOTO( $\text{TOP}(\omega_{\text{part}}), X$ ) @  $i$ 
  add PUSH( $\omega_{\text{part}}, \sigma_{\text{next}}$ ) to  $K^j$ 

return  $K^j$ 

```

Reduction

Nondeterministic reduction also loops over the partition, and does reduction on each part separately, taking the union of the results. Also note that the original set of stacks is included in the final reduction result, since it is always possible that some stack has finished reducing and should shift next.

function `REDUCE`(K)

```

for all  $\omega_{\text{part}} \in \text{TOP-PARTITION}(K)$  do
   $q_{\text{top}}@i := \text{TOP}(\omega_{\text{part}})$ 
  for all  $(A \rightarrow B_1 \cdots B_d) \in q_{\text{top}}$  do
     $\omega_{\text{red}} := \text{POP}(\omega_{\text{part}})$ 
    add REDUCE(SHIFT( $\omega_{\text{red}}, A@i$ )) to  $K$ 

return  $K$ 

```

Grammars with Empty Productions

The GLR algorithm as it is described in this chapter cannot correctly handle all grammars with ϵ -rules. This is a well-known problem for GLR parsers, and there are two main solutions. One possibility is of course to transform the grammar into ϵ -free form (Hopcroft et al. 2006). Another possibility is to modify the GLR algorithm, possibly together with a modified LR table (Nozohoor-Farshi 1991, Nederhof and Sarbo 1996, Aycock et al. 2001, Aycock and Horspool 2002, Scott and Johnstone 2006, Scott et al. 2007).

4.6.4 Optimized GLR Parsing

Each stack that survives in the previous set-based algorithm corresponds to a possible parse tree. But since there can be exponentially many parse trees, this means that the algorithm is exponential in the length of the input.

The problem is the data structure—a set of stacks does not take into account that parallel stacks often have several parts in common. Tomita (1985, 1988) suggested to store a set of stacks as a directed acyclic graph (calling it a *graph-structured stack*), which together with suitable algorithms make GLR parsing polynomial in the length of the input.

The only things we have to do is to reimplement the five operations POP, PUSH, TOP-PARTITION, TOP, and ($\cup$); the functions LR, SHIFT, and REDUCE all stay the same. We represent a graph-structured stack as a pair $G : T$, where G is a directed graph over indexed states, and T is a subset of the nodes in G that constitute the current stack tops. Assuming that the graph is represented by a set of directed edges $\sigma^j \rightarrow \sigma$, the operations can be implemented as follows:

$$\begin{aligned} \text{POP}(G : T) &= G : \sigma^j \mid \sigma \in T, \sigma^j \rightarrow \sigma \in G \\ \text{PUSH}(G : T, \sigma) &= G \cup \begin{matrix} \cdot \\ \sigma^j \end{matrix} \rightarrow \sigma \mid \sigma^j \in T : \{\sigma\} \\ \text{TOP-PARTITION}(G : T) &= \{ G : \{\sigma\} \mid \sigma \in T \} \\ \text{TOP}(G : \{\sigma\}) &= \sigma \\ (G_1 : T_1) \cup (G_2 : T_2) &= (G_1 \cup G_2) : (T_1 \cup T_2) \end{aligned}$$

The initial stack in the top-level LR function is still conceptually a singleton set, but will be encoded as a graph-structured stack $\emptyset : \{q_{\text{init}}@0\}$. Note that for the graph-structured version to be correct, we need the LR states to be indexed. Otherwise the graph will merge all nodes having the same LR state, regardless of where in the input it is recognized.

The graph-structured stack operations never remove edges from the graph, only add new edges. This means that it is possible to implement GLR parsing using a global graph, where the only thing that is passed around is the set T of stack tops.

Tabular GLR Parsing

The astute reader might have noticed the similarity between the graph-structured stack and the chart in Section 4.5: The graph (chart) is a global set of edges (items), to which edges (items) are added during parsing, but never removed. It should therefore be possible to reformulate GLR parsing as a tabular algorithm. For this we need two inference rules, corresponding to SHIFT and REDUCE, and one axiom corresponding to the initial stack. This tabular GLR algorithm is described by Nederhof and Satta (2004b).

4.7 Constraint-Based Grammars

This section introduces a simple form of constraint-based grammar, or unification grammar, which for more than two decades has constituted a widely adopted class of formalisms in computational linguistics.

4.7.1 Overview

A key characteristic of constraint-based formalisms is the use of feature terms (sets of attribute–value pairs) for the description of linguistic units, rather than atomic categories as in CFGs. Feature terms can be nested: their values can be either atomic symbols or feature terms. Furthermore, they are partial (underspecified) in the sense that new information may be added as long as it is compatible with old

information. The operation for merging and checking compatibility of feature constraints is usually formalized as *unification*. Some formalisms, such as PATR (Shieber et al. 1983, Shieber 1986) and Regulus (Rayner et al. 2006), are restricted to simple unification (of conjunctive terms), while others such as LFG (Kaplan and Bresnan 1982) and HPSG (Pollard and Sag 1994) allow disjunctive terms, sets, type hierarchies, or other extensions. In sum, feature terms have proved to be an extremely versatile and powerful device for linguistic description. One example of this is unbounded dependency, as illustrated by examples (4.1)–(4.3) in Section 4.1, which can be handled entirely within the feature system by the technique of gap threading (Karttunen 1986).

Several constraint-based formalisms are phrase-structure-based in the sense that each rule is factored in a phrase-structure backbone and a set of constraints that specify conditions on the feature terms associated with the rule (e.g., PATR, Regulus, CLE, HPSG, LFG, and TAG, though the latter uses certain tree-building operations instead of rules). Analogously, when parsers for constraint-based formalisms are built, the starting-point is often a phrase-structure parser that is augmented to handle feature terms. This is also the approach we shall follow here.

4.7.2 Unification

We make use of a constraint-based formalism with a context-free backbone and restricted to simple unification (of conjunctive terms), thus corresponding to PATR (Shieber et al. 1983, Shieber 1986). A grammar rule in this formalism can be seen as an ordered pair of a production $X_0 \rightarrow X_1 \cdot \cdot \cdot X_d$ and a set of equational constraints over the feature terms of types $X_0, \dots, X_d$. A simple example of a rule, encoding agreement between the determiner and the noun in a noun phrase, is the following:

$$\begin{aligned}
 X_0 &\rightarrow X_1 X_2 \\
 X_0 \text{ category} &= \text{NP} \\
 \cdot X_1 \text{ category} &= \text{Det} \\
 \cdot X_2 \text{ category} &= \text{Noun} \\
 \cdot X_1 \text{ agreement} &= \cdot X_2 \text{ agreement}
 \end{aligned}$$

Any such rule description can be represented as a phrase-structure rule where the symbols consist of feature terms. Below is a feature term rule corresponding to the previous rule (where $\overline{1}$ indicates identity between the associated elements):

$$\begin{array}{c} \overline{1} \\ \text{category : NP} \end{array} \rightarrow \begin{array}{c} \overline{1} \\ \text{category : Det} \\ \text{agreement } \overline{1} \end{array} \begin{array}{c} \overline{1} \\ \text{category : Noun} \\ \text{agreement } \overline{1} \end{array}$$

The basic operation on feature terms is *unification*, which determines if two terms are compatible by merging them to the most general term compatible with both. As an example, the unification $A \text{ h } B$ of the terms $A = \text{agreement number : plural}$ and $B = \text{agreement gender : } \Sigma$ succeeds with the result:

$$A \text{ h } B = \begin{array}{c} \Sigma \quad \Sigma \quad \Sigma \quad \Sigma \\ \text{agreement : } \\ \text{gender : neutr} \\ \text{number : plural} \end{array}$$

However, neither A nor $A \sqcap B$ can be unified with

$$C = \begin{bmatrix} \text{agreement} : \text{number} : \text{singular} \end{bmatrix}$$

since the atomic values *plural* and *singular* are distinct. The semantics of feature terms including the unification algorithm is described by Pereira and Shieber (1984), Kasper and Rounds (1986), and Shieber

(1992). The unification of feature terms is an extension of Robinson's unification algorithm for first-order terms (Robinson 1965). More advanced grammar formalisms such as HPSG and LFG use further extensions of feature terms, such as type hierarchies and disjunction.

4.7.3 Tabular Parsing with Unification

Basically, the tabular parsers in Section 4.4 as well as the GLR parsers in Section 4.6 can be adapted to constraint-based grammar by letting the symbols in the grammar rules be feature terms instead of atomic nonterminal symbols (Shieber 1985b, Tomita 1987, Nakazawa 1991, Samuelsson 1994). For example, an item in tabular parsing then still has the form $i, j: A \rightarrow \alpha \cdot \beta$, where A is a feature term and α, β are sequences of feature terms.

A problem in tabular parsing with constraint-based grammar as opposed to CFG is that the item-redundancy test involves comparing complex feature terms instead of testing for equality between atomic symbols. For this reason, we need to make sure that no previously added item *subsumes* a new item to be added (Shieber 1985a, Pereira and Shieber 1987). Informally, an item e subsumes another item e^j if e contains a subset of the information in e^j . Since the input positions i, j are always fully instantiated, this amounts to checking if the feature terms in the dotted rule of e subsumes the corresponding feature terms in e^j . The rationale for using this test is that we are only interested in adding edges that are less specific than the old ones, since everything we could do with a more specific edge, we can also do with a more general one.

The algorithm for implementing the deduction engine, presented in Section 4.5.1, only needs minor modifications to work on unification-based grammars: (1) instead of checking that the new item e is contained in the chart, we check that there is an item in the chart that subsumes e , and (2) instead of testing whether two items e_j and e_j^j matches, we try to perform the unification $e_j \mathbf{h} e_j^j$.

However, subsumption testing is not always sufficient for correct and efficient tabular parsing, since tabular CFG-based parsers are not fully specified in the order in which ambiguities are discovered (Lavie and Rosé 2004). Unification grammars may contain rules that lead to the prediction of ever more specific feature terms that do not subsume each other, thereby resulting in infinite sequences of predictions. This kind of problem occurs in natural language grammars when keeping lists of, say, subcategorized constituents or gaps to be found. In logic programming, the *occurs check* is used for circumventing a corresponding circularity problem. In constraint-based grammar, Shieber (1985b) introduced the notion of *restriction* for the same purpose. A restrictor removes those portions of a feature term that could potentially lead to non-termination. This is in general done by replacing those portions with free (newly instantiated) variables, which typically removes some coreference. The purpose of restriction is to ensure that terms to be predicted are only instantiated to a certain depth, such that terms will eventually subsume each other.

4.8 Issues in Parsing

In the light of the previous exposition, this section reexamines the three fundamental challenges of parsing discussed in Section 4.1.

4.8.1 Robustness

Robustness can be seen as the ability to deal with input that somehow does not conform to what is normally expected (Menzel 1995). In grammar-driven parsing, it is natural to take “expected” input to correspond to those strings that are in the formal language $L(G)$ generated by the grammar G . However, as discussed in Section 4.1, a natural language parser will always be exposed to some amount of input

that is not in $L(G)$. One source of this problem is *undergeneration*, which is caused by a lack of coverage

of G relative to the natural language L . Another problem is that the input may contain errors; in other words, that it may be *ill-formed* (though the distinction between well-formed and ill-formed input is by no means clear-cut). But regardless of why the input is not in $L(G)$, it is usually desirable to try to recover as much meaningful information from it as possible, rather than returning no result at all. This is the problem of *robustness*, whose basic notion is to always return *some* analysis of the input. In a stronger sense, robustness means that small deviations from the expected input will only cause small impairments of the parse result, whereas large deviations may cause large impairments. Hence, robustness in this stronger sense amounts to *graceful degradation*.

Clearly, robustness requires methods that sacrifice something from the traditional ideal of recovering complete and exact parses using a linguistically motivated grammar. To avoid the situation where the parser can only stop and report failure in analyzing the input, one option is to *relax* some of the grammatical constraints in such a way that a (potentially) ungrammatical sentence obtains a complete analysis (Jensen and Heidorn 1983, Mellish 1989). Put differently, by relaxing some constraints, a certain amount of *overgeneration* is achieved relative to the original grammar, and this is then hopefully sufficient to account for the input. The key problem of this approach is that, as the number of errors grows, the number of relaxation alternatives that are compatible with analyses of the whole input may explode, and that the search for a best solution is therefore very difficult to control.

One can then instead focus on the design of the *grammar*, making it less rich in the hope that this will allow for processing that is less brittle. The amount of information contained in the structural representations yielded by the parser is usually referred to as a distinction between *deep parsing* and *shallow parsing* (somewhat misleadingly, as this distinction does not necessarily refer to different parsing methods per se, but rather to the syntactic representations used). Deep parsing systems typically capture long-distance dependencies or predicate–argument relations directly, as in LFG, HPSG, or CCG (compare Section 4.2.4). In contrast, shallow parsing makes use of more skeletal representations. An example of this is Constraint Grammar (Karlsson et al. 1995). This works by first assigning all possible part-of-speech and syntactic labels to all words. It then applies pattern-matching rules (constraints) to disambiguate the labels, thereby reducing the number of parses. The result constitutes a dependency structure in the sense that it only provides relations between words, and may be ambiguous in that the identities of dependents are not fully specified.

A distinction which is sometimes used more or less synonymously with deep and shallow parsing is that between *full parsing* and *partial parsing*. Strictly speaking, however, this distinction refers to the degree of completeness of the analysis with respect to a given target representation. Thus, partial parsing is often used to denote an initial, surface-oriented analysis (“almost parsing”), in which certain decisions, such as attachments, are left for subsequent processing. A radical form of partial parsing is *chunk parsing* (Abney 1991, 1997), which amounts to finding boundaries between basic elements, such as non-recursive clauses or low-level phrases, and analyzing each of these elements using a finite-state grammar. Higher-level analysis is then left for processing by other means. One of the earliest approaches to partial parsing was *Fidditch* (Hindle 1989, 1994). A key idea of this approach is to leave constituents whose roles cannot be determined unattached, thereby always providing exactly one analysis for any given sentence. Another approach is *supertagging*, introduced by Bangalore and Joshi (1999) for the LTAG formalism as a means to reduce ambiguity by associating lexical items with rich descriptions (supertags) that impose complex constraints in a local context, but again without itself deriving a syntactic analysis. Supertagging has also been successfully applied within the CCG formalism (Clark and Curran 2004).

A second option is to sacrifice completeness with respect to covering the entire *input*, by parsing only fragments that are well-formed according to the grammar. This is sometimes referred to as *skip parsing*. Partial parsing is a means to achieve this, since leaving a fragment unattached may just as well be seen as a way of skipping that fragment. A particularly important case for skip parsing is noisy input, such as written text containing errors or output from a speech recognizer. (A word error rate around 20%–40% is by no means unusual in recognition of spontaneous speech; see Chapter 15.) For the parsing of spoken

language in conversational systems, it has long been commonplace to use pattern-matching rules that

trigger on domain-dependent subsets of the input (Ward 1989, Jackson et al. 1991, Boye and Wirén 2008). Other approaches have attempted to render deep parsing methods robust, usually by trying to connect the maximal subset of the original input that is covered by the grammar. For example, GLR* (Lavie 1996, Lavie and Tomita 1996), an extension of GLR (Section 4.6.3), can parse all subsets of the input that are licensed by the grammar by being able to skip over any words. Since many parsable subsets of the original input must then be analyzed, the amount of ambiguity is greatly exacerbated. To control the search space, GLR* makes use of statistical disambiguation similar to a method proposed by Carroll (1993) and Briscoe and Carroll (1993), where probabilities are associated directly with the actions in the pre-compiled LR parsing table (a method that in turn is an instance of the conditional history-based models discussed in Chapter 11). Other approaches in which subsets of the input can be parsed are Rosé and Lavie (2001), van Noord et al. (1999), and Kasper et al. (1999).

A third option is to sacrifice the traditional notion of *constructive parsing*, that is, analyzing sentences by building syntactic representations imposed by the rules of a grammar. Instead one can use *eliminative parsing*, which works by initially setting up a maximal set of conditions, and then gradually reducing analyses that are illegal according to a given set of constraints, until only legal analyses remain. Thus, parsing is here viewed as a constraint satisfaction problem (or, put differently, as disambiguation), in which the set of constraints guiding the process corresponds to the grammar. Examples of this kind of approach are Constraint Grammar (Karlsson et al. 1995) and the system of Foth and Menzel (2005).

4.8.2 Disambiguation

The dual problem of undergeneration is that the parser produces superfluous analyses, for example, in the form of *massive ambiguity*, as illustrated in Section 4.1. Ultimately, we would like not just *some* analysis (robustness), but rather *exactly one* (disambiguation). Although not all information needed for disambiguation (such as contextual constraints) may be available during parsing, some pruning of the search space is usually possible and desirable. The parser may then pass on the *best* analyses, if not a single one, to the next level of processing. A related problem, and yet another source of superfluous analyses, is that the grammar might be incomplete not only in the sense of undergeneration, but also by licensing constructions that do not belong to the natural language *L*. This problem is known as *overgeneration* or *leakage*, by reference to Sapir's famous statement that "[a]ll grammars leak" (Sapir 1921, p. 39).

A basic observation is that, although a general grammar will allow a large number of analyses of almost any nontrivial sentence, most of these analyses will be extremely implausible in the context of a particular domain. A simple approach that was pursued early on was then to code a new, specialized *semantic grammar* for each domain (Burton 1976, Hendrix et al. 1978).^{*} A more advanced alternative is to tune the parser and/or grammar for each new domain. Grishman et al. (1984), Samuelsson and Rayner (1991), and Rayner et al. (2000) make use of a method known as *grammar specialization*, which takes advantage of actual rule usage in a particular domain. This method is based on the observation that, in a given domain, certain groups of grammar rules tend to combine frequently in some ways but not in others. On the basis of a sufficiently large corpus parsed by the original grammar, it is then possible to identify common combinations of rules of a (unification) grammar and to collapse them into single "macro" rules. The result is a specialized grammar, which, compared to the original grammar, has a larger number of rules but a simpler structure, reducing ambiguity and allowing very fast processing using an LR parser. Another possibility is to use a hybrid method to rank a set of analyses according to their likelihood in the domain, based on data from supervised training (Rayner et al. 2000, Toutanova et al. 2002). Clearly, disambiguation leads naturally in one way or another to the application of statistical inference; for a systematic exposition of this, we refer to Chapter 11.

* Note that this early approach can be seen as a text-oriented and less robust variant of the domain-dependent pattern-matching systems of Ward (1989) and others aimed at spoken language, referred to in Section 4.8.1.

4.8.3 Efficiency

Theoretical Time Complexity

The worst-case time complexity for parsing with CFG is cubic, $O(n^3)$, in the length of the input sentence. This can most easily be seen for the algorithm `CKY()` in Section 4.3. The main part consists of three nested loops, all ranging over $O(n)$ input positions, giving cubic time complexity. This is not changed by the `UNARY-CKY()` algorithm. However, if we add inner loops for handling long right-hand sides, as discussed in Section 4.3.3, the complexity increases to $O(n^{d+1})$, where d is the length of the longest right-hand side in the grammar.

The time complexities of the tabular algorithms in Section 4.4 are also cubic, since using dotted rules constitutes an implicit transformation of the grammar into binary form. In general, assuming that we have a decent implementation of the deduction engine, the time complexity of a deductive algorithm is the complexity of the most complex inference rule. In our case this is the combine rule (4.10), which contains three variables i, j, k ranging over $O(n)$ input positions.

The worst-case time complexity of the optimized GLR algorithm, as formulated in Section 4.6.4, is $O(n^{d+1})$. This is because reduction pops the stack d times, for a rule with right-hand side length d . By binarizing the stack reductions it is possible to obtain cubic time complexity for GLR parsing (Kipps 1991, Nederhof and Satta 1996, Scott et al. 2007).

If the CFG is lexicalized, as mentioned in Section 4.2.4, the time complexity of parsing becomes $O(n^5)$ rather than cubic. The reason for this is that the cubic parsing complexity also depends on the grammar size, which for a bilexical CFG depends quadratically on the size of the lexicon. And after filtering out the grammar rules that do not have a realization in the input sentence, we obtain a complexity of $O(n^2)$ (for the grammar size) multiplied by $O(n^3)$ (for context-free parsing). Eisner and Satta (1999) and Eisner (2000) provide an $O(n^4)$ algorithm for bilexical CFG, and an $O(n^3)$ algorithm for a common restricted class of lexicalized grammars.

Valiant (1975) showed that it is possible to transform the CKY algorithm into the problem of Boolean matrix multiplication (BMM), for which there are sub-cubic algorithms. Currently, the best BMM algorithm is approximately $O(n^{2.376})$ (Coppersmith and Winograd 1990). However, these sub-cubic algorithms all involve large constants making them inefficient in practice. Furthermore, since BMM can be reduced to context-free parsing (Lee 2002), there is not much hope in finding practical parsing algorithms with sub-cubic time complexity.

As mentioned in Section 4.2.4, MCS grammar formalisms all have polynomial parse time complexity. More specifically, TAG and CCG have $O(n^6)$ time complexity (Vijay-Shanker and Weir 1993), whereas for LCFRS, MCFG, and RCG the exponent depends on the complexity of the grammar (Satta 1992).

In general, adding feature terms and unification to a phrase-structure backbone makes the resulting formalism undecidable. In practice, however, conditions are often placed on the phrase-structure backbone and/or possible feature terms to reduce complexity (Kaplan and Bresnan 1982, p. 266; Pereira and Warren 1983, p. 142), sometimes even to the effect of retaining polynomial parsability (Joshi 1997). For a general exposition of computational complexity in connection with linguistic theories, see Barton et al. (1987).

Practical Efficiency

The complexity results above represent theoretical worst cases, which in actual practice may occur only under very special circumstances. Hence, to assess the practical behavior of parsing algorithms, empirical evaluations are more informative. As an illustration of this, in a comparison of three unification-based parsers using a wide-coverage grammar of English, Carroll (1993) found parsing times for exponential-time algorithms to be approximately quadratic in the length of the input for sentence lengths of 1–30 words.

Early work on empirical parser evaluation, such as Pratt (1975), Slocum (1981), Tomita (1985), Wirén

(1987) and Billot and Lang (1989), focused on the behavior of specific algorithms. However, reliable

comparisons require that the same grammars and test data are used across different evaluations. Increasingly, the availability of common infrastructure in the form of grammars, treebanks, and test suites has facilitated this, as illustrated by Carroll (1994), van Noord (1997), Oepen et al. (2000), Oepen and Carroll (2002), and Kaplan et al. (2004), among others. A more difficult problem is that reliable comparisons also require that parsing times can be normalized across different implementations and computing platforms. One way of trying to handle this complication would be to have standard implementations of reference algorithms in all implementation languages of interest, as suggested by Moore (2000).

4.9 Historical Notes and Outlook

With the exception of machine translation, parsing is probably the area with the longest history in natural language processing. Victor Yngve has been credited with describing the first method for parsing, conceived of as one component of a system for machine translation, and proceeding bottom-up (Yngve 1955). Subsequently, top-down algorithms were provided by, among others, Kuno and Oettinger (1962). Another early approach was the Transformation and Discourse Analysis Project (TDAP) of Zellig Harris 1958–1959, which in effect used cascades of finite-state automata for parsing (Harris 1962; Joshi and Hopely 1996).

During the next decade, the focus shifted to parsing algorithms for context-free grammar. In 1960, John Cocke invented the core dynamic-programming parser that was independently generalized and formalized by Kasami (1965) and Younger (1967), thus evolving into the CKY algorithm. This allowed for parsing in cubic time with grammars in CNF. Although Cocke's original algorithm was never published, it remains a highly significant achievement in the history of parsing (for sources to this, see Hays 1966 and Kay 1999). In 1968, Earley then presented the first algorithm for parsing with general CFG in no worse than cubic time (Earley 1970). In independent work, Kay (1967, 1973, 1986) and Kaplan (1973) generalized Cocke's algorithm into what they coined chart parsing. A key idea of this is to view tabular parsing algorithms as instances of a general algorithm schema, with specific parsing algorithms arising from different instantiations of inference rules and the agenda (see also Thompson 1981, 1983, Wirén 1987).

However, with the growing dominance of Transformational Grammar, particularly with the *Aspects* ("Standard Theory") model introduced by Chomsky (1965), there was a diminished interest in context-free phrase-structure grammar. On the other hand, Transformational Grammar was not itself amenable to parsing in any straightforward way. The main reason for this was the inherent directionality of the transformational component, in the sense that it maps from deep structure to surface word string.

A solution to this problem was the development of Augmented Transition Networks (ATNs), which started in the late 1960s and which became the dominating framework for natural language processing during the 1970s (Woods et al. 1972, Woods 1970, 1973). Basically, the appeal of the ATN was that it constituted a formalism of the same power as Transformational Grammar, but one whose operational claims could be clearly stated, and which provided an elegant (albeit procedural) way of linking the surface structure encoded by the network path with the deep structure built up in registers.

Beginning around 1975, there was a revival of interest in phrase-structure grammars (Joshi et al. 1975, Joshi 1985), later augmented with complex features whose values were typically matched using unification (Shieber 1986). One reason for this revival was that some of the earlier arguments against the use of CFG had been refuted, resulting in several systematically restricted formalisms (see Section 4.2.4). Another reason was a movement toward declarative (constraint-based) grammar formalisms that typically used a phrase-structure backbone, and whose parsability and formal properties could be rigorously analyzed. This allowed parsing to be formulated in ways that abstracted from implementational detail, as demonstrated most elegantly in the parsing-as-deduction paradigm (Pereira and Warren 1983).

Another development during this time was the generalization of Knuth's deterministic LR parsing algorithm (Knuth 1965) to handling nondeterminism (ambiguous CFGs), leading to the notion of GLR

parsing (Lang 1974, Tomita 1985). Eventually, the relation of this framework to tabular (chart) parsing

was also illuminated (Nederhof and Satta 1996, 2004b). Finally, in contrast to the work based on phrase-structure grammar, there was a renewed interest in more restricted and performance-oriented notions of parsing, such as finite-state parsing (Church 1980, Ejerhed 1988) and deterministic parsing (Marcus 1980, Shieber 1983).

In the late 1980s and during the 1990s, two interrelated developments were particularly apparent: on the one hand, an interest in robust parsing, motivated by an increased involvement with unrestricted text and spontaneous speech (see Section 4.8.1), and on the other hand the revival of empiricism, leading to statistics-based methods being applied both on their own and in combination with grammar-driven parsing (see Chapter 11). These developments have continued during the first decade in the new millenium, along with a gradual closing of the divide between grammar-driven and statistical methods (Nivre 2002, Baldwin et al. 2007).

In sum, grammar-driven parsing is one of the oldest areas within natural language processing, and one whose methods continue to be a key component of much of what is carried out in the field. Grammar-driven approaches are essential when the goal is to achieve the precision and rigor of deep parsing, or when annotated corpora for supervised statistical approaches are unavailable. The latter situation holds both for the majority of the world's languages and frequently when systems are to be engineered for new application domains. But also in shallow and partial parsing, some of the most successful systems in terms of accuracy and efficiency are rule based. However, the best performing broad-coverage parsers for theoretical frameworks such as CCG, HPSG, LFG, TAG, and dependency grammar increasingly use statistical components for preprocessing (e.g., tagging) and/or postprocessing (by ranking competing analyses for the purpose of disambiguation). Thus, although grammar-driven approaches remain a basic framework for syntactic parsing, it appears that we can continue to look forward to an increasingly symbiotic relationship between grammar-driven and statistical methods.

Acknowledgments

We want to thank the reviewers, Alon Lavie and Mark-Jan Nederhof, for detailed and constructive comments on an earlier version of this chapter. We also want to thank Joakim Nivre for helpful comments and for discussions about the organization of the two parsing chapters (Chapters 4 and 11).

References

- Abney, S. (1991). Parsing by chunks. In R. Berwick, S. Abney, and C. Tenny (Eds.), *Principle-Based Parsing*, pp. 257–278. Kluwer Academic Publishers, Dordrecht, the Netherlands.
- Abney, S. (1997). Part-of-speech tagging and partial parsing. In S. Young and G. Bloothoof (Eds.), *Corpus-Based Methods in Language and Speech Processing*, pp. 118–136. Kluwer Academic Publishers, Dordrecht, the Netherlands.
- Aho, A., M. Lam, R. Sethi, and J. Ullman (2006). *Compilers: Principles, Techniques, and Tools* (2nd ed.). Addison-Wesley, Reading, MA.
- Aycock, J. and N. Horspool (2002). Practical Earley parsing. *The Computer Journal* 45(6), 620–630.
- Aycock, J., N. Horspool, J. Janoušek, and B. Melichar (2001). Even faster generalized LR parsing. *Acta Informatica* 37(9), 633–651.
- Backus, J. W., F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger (1963). Revised report on the algorithm language ALGOL 60. *Communications of the ACM* 6(1), 1–17.

Baldwin, T., M. Dras, J. Hockenmaier, T. H. King, and G. van Noord (2007). The impact of deep linguistic processing on parsing technology. In *Proceedings of the 10th International Conference on Parsing Technologies, IWPT'07*, Prague, Czech Republic, pp. 36–38.

- Bangalore, S. and A. K. Joshi (1999). Supertagging: An approach to almost parsing. *Computational Linguistics* 25(2), 237–265.
- Bar-Hillel, Y., M. Perles, and E. Shamir (1964). On formal properties of simple phrase structure grammars. In Y. Bar-Hillel (Ed.), *Language and Information: Selected Essays on Their Theory and Application*, Chapter 9, pp. 116–150. Addison-Wesley, Reading, MA.
- Barton, G. E., R. C. Berwick, and E. S. Ristad (1987). *Computational Complexity and Natural Language*. MIT Press, Cambridge, MA.
- Billot, S. and B. Lang (1989). The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics, ACL'89*, Vancouver, Canada, pp. 143–151.
- Boullier, P. (2004). Range concatenation grammars. In H. Bunt, J. Carroll, and G. Satta (Eds.), *New Developments in Parsing Technology*, pp. 269–289. Kluwer Academic Publishers, Dordrecht, the Netherlands.
- Boye, J. and M. Wirén (2008). Robust parsing and spoken negotiative dialogue with databases. *Natural Language Engineering* 14(3), 289–312.
- Bresnan, J. (2001). *Lexical-Functional Syntax*. Blackwell, Oxford, U.K.
- Briscoe, T. and J. Carroll (1993). Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics* 19(1), 25–59.
- Burton, R. R. (1976). Semantic grammar: An engineering technique for constructing natural language understanding systems. BBN Report 3453, Bolt, Beranek, and Newman, Inc., Cambridge, MA.
- Carpenter, B. (1992). *The Logic of Typed Feature Structures*. Cambridge University Press, New York.
- Carroll, J. (1993). Practical unification-based parsing of natural language. PhD thesis, University of Cambridge, Cambridge, U.K. Computer Laboratory Technical Report 314.
- Carroll, J. (1994). Relating complexity to practical performance in parsing with wide-coverage unification grammars. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics, ACL'94*, Las Cruces, NM, pp. 287–294.
- Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory* 2(3), 113–124.
- Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control* 2(2), 137–167.
- Chomsky, N. (1965). *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA.
- Church, K. W. (1980). On memory limitations in natural language processing. Report MIT/LCS/TM-216, Massachusetts Institute of Technology, Cambridge, MA.
- Church, K. W. and R. Patil (1982). Coping with syntactic ambiguity or how to put the block in the box on the table. *Computational Linguistics* 8(3–4), 139–149.
- Clark, S. and J. R. Curran (2004). The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of the 20th International Conference on Computational Linguistics, COLING'04*, Geneva, Switzerland.
- Coppersmith, D. and S. Winograd (1990). Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation* 9(3), 251–280.
- Daniels, M. W. and D. Meurers (2004). A grammar formalism and parser for linearization-based HPSG. In *Proceedings of the 20th International Conference on Computational Linguistics, COLING'04*, Geneva, Switzerland, pp. 169–175.

- de Groote, P. (2001). Towards abstract categorial grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics, ACL'01*, Toulouse, France.
- Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM* 13(2), 94–102.
- Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In H. Bunt and A. Nijholt (Eds.), *New Developments in Natural Language Parsing*. Kluwer Academic Publishers, Dordrecht, the Netherlands.

- Eisner, J. and G. Satta (1999). Efficient parsing for bilexical context-free grammars and head automata grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, ACL'99*, pp. 457–464.
- Ejerhed, E. (1988). Finding clauses in unrestricted text by finitary and stochastic methods. In *Proceedings of the Second Conference on Applied Natural Language Processing*, Austin, TX, pp. 219–227.
- Foth, K. and W. Menzel (2005). Robust parsing with weighted constraints. *Natural Language Engineering* 11(1), 1–25.
- Gazdar, G., E. Klein, G. Pullum, and I. Sag (1985). *Generalized Phrase Structure Grammar*. Basil Blackwell, Oxford, U.K.
- Grishman, R., N. T. Nhan, E. Marsh, and L. Hirschman (1984). Automated determination of sublanguage syntactic usage. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd Annual Meeting of the Association for Computational Linguistics, COLING-ACL'84*, Stanford, CA, pp. 96–100.
- Harris, Z. S. (1962). *String Analysis of Sentence Structure*. Mouton, Hague, the Netherlands.
- Hays, D. G. (1966). Parsing. In D. G. Hays (Ed.), *Readings in Automatic Language Processing*, pp. 73–82. American Elsevier Publishing Company, New York.
- Hendrix, G. G., E. D. Sacerdoti, and D. Sagalowicz (1978). Developing a natural language interface to complex data. *ACM Transactions on Database Systems* 3(2), 105–147.
- Hindle, D. (1989). Acquiring disambiguation rules from text. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics, ACL'89*, Vancouver, Canada, pp. 118–125.
- Hindle, D. (1994). A parser for text corpora. In A. Zampolli (Ed.), *Computational Approaches to the Lexicon*. Oxford University Press, New York.
- Hopcroft, J., R. Motwani, and J. Ullman (2006). *Introduction to Automata Theory, Languages, and Computation* (3rd ed.). Addison-Wesley, Boston, MA.
- Jackson, E., D. Appelt, J. Bear, R. Moore, and A. Podlozny (1991). A template matcher for robust NL interpretation. In *Proceedings of the Workshop on Speech and Natural Language, HLT'91*, Pacific Grove, CA, pp. 190–194.
- Jensen, K. and G. E. Heidorn (1983). The fitted parse: 100% parsing capability in a syntactic grammar of English. In *Proceedings of the First Conference on Applied Natural Language Processing*, Santa Monica, CA, pp. 93–98.
- Joshi, A. K. (1997). Parsing techniques. In R. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, and V. Zue (Eds.), *Survey of the State of the Art in Human Language Technology*, pp. 351–356. Cambridge University Press, Cambridge, MA.
- Joshi, A. K. (1985). How much context-sensitivity is necessary for characterizing structural descriptions – tree adjoining grammars. In D. Dowty, L. Karttunen, and A. Zwicky (Eds.), *Natural Language Processing: Psycholinguistic, Computational and Theoretical Perspectives*, pp. 206–250. Cambridge University Press, New York.
- Joshi, A. K. and P. Hopely (1996). A parser from antiquity: A nearly application of finite state transducers to natural language parsing. *Natural Language Engineering* 2(4), 291–294.
- Joshi, A. K., L. S. Levy, and M. Takahashi (1975). Tree adjunct grammars. *Journal of Computer and System Sciences* 10(1), 136–163.

- Joshi, A. K. and Y. Schabes (1997). Tree-adjoining grammars. In G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages. Vol 3: Beyond Words*, Chapter 2, pp. 69–123. Springer-Verlag, Berlin.
- Kaplan, R. and J. Bresnan (1982). Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan (Ed.), *The Mental Representation of Grammatical Relations*, pp. 173–281. MIT Press, Cambridge, MA.
- Kaplan, R. M. (1973). A general syntactic processor. In R. Rustin (Ed.), *Natural Language Processing*, pp. 193–241. Algorithmics Press, New York.

- Kaplan, R. M., S. Riezler, T. H. King, J. T. Maxwell III, A. Vasserman, and R. S. Crouch (2004). Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT- NAACL'04*, Boston, MA, pp. 97–104.
- Kaplan, R. M. and A. Zaenen (1995). Long-distance dependencies, constituent structure, and functional uncertainty. In R. M. Kaplan, M. Dalrymple, J. T. Maxwell, and A. Zaenen (Eds.), *Formal Issues in Lexical-Functional Grammar*, Chapter 3, pp. 137–165. CSLI Publications, Stanford, CA.
- Karlsso, F., A. Voutilainen, J. Heikkilä, and A. Anttila (Eds.) (1995). *Constraint Grammar. A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter, Berlin, Germany.
- Karttunen, L. (1986). D-PATR: A development environment for unification-based grammars. In *Proceedings of 11th International Conference on Computational Linguistics, COLING'86*, Bonn, Germany.
- Karttunen, L. and A. M. Zwicky (1985). Introduction. In D. Dowty, L. Karttunen, and A. Zwicky (Eds.), *Natural Language Processing: Psycholinguistic, Computational and Theoretical Perspectives*, pp. 1–25. Cambridge University Press, New York.
- Kasami, T. (1965). An efficient recognition and syntax algorithm for context-free languages. Technical Report AFCLR-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.
- Kasper, R. T. and W. C. Rounds (1986). A logical semantics for feature structures. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics, ACL'86*, New York, pp. 257–266.
- Kasper, W., B. Kiefer, H. U. Krieger, C. J. Rupp, and K. L. Worm (1999). Charting the depths of robust speech parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, ACL'99*, College Park, MD, pp. 405–412.
- Kay, M. (1967). Experiments with a powerful parser. In *Proceedings of the Second International Conference on Computational Linguistics [2ème conférence internationale sur le traitement automatique des langues]*, COLING'67, Grenoble, France.
- Kay, M. (1973). The MIND system. In R. Rustin (Ed.), *Natural Language Processing*, pp. 155–188. Algorithmics Press, New York.
- Kay, M. (1986). Algorithm schemata and data structures in syntactic processing. In B. Grosz, K. S. Jones, and B. L. Webber (Eds.), *Readings in Natural Language Processing*, pp. 35–70, Morgan Kaufmann Publishers, Los Altos, CA. Originally published as Report CSL-80-12, Xerox PARC, Palo Alto, CA, 1980.
- Kay, M. (1989). Head-driven parsing. In *Proceedings of the First International Workshop on Parsing Technologies, IWPT'89*, Pittsburgh, PA.
- Kay, M. (1999). Chart translation. In *Proceedings of the MT Summit VII*, Singapore, pp. 9–14.
- Kipps, J. R. (1991). GLR parsing in time $O(n^3)$. In M. Tomita (Ed.), *Generalized LR Parsing*, Chapter 4, pp. 43–59. Kluwer Academic Publishers, Boston, MA.
- Knuth, D. E. (1965). On the translation of languages from left to right. *Information and Control* 8, 607–639.
- Kuno, S. and A. G. Oettinger (1962). Multiple-path syntactic analyzer. In *Proceedings of the IFIP Congress*, Munich, Germany, pp. 306–312.
- Lang, B. (1974). Deterministic techniques for efficient non-deterministic parsers. In J. Loeckx (Ed.), *Proceedings of the Second Colloquium on Automata, Languages and Programming*, Saarbrücken, Germany, Volume 14 of LNCS, pp. 255–269. Springer-Verlag, London, U.K.

- Lavie, A. (1996). GLR*: A robust parser for spontaneously spoken language. In *Proceedings of the ESSLLI'96 Workshop on Robust Parsing*, Prague, Czech Republic.
- Lavie, A. and C. P. Rosé (2004). Optimal ambiguity packing in context-free parsers with interleaved unification. In H. Bunt, J. Carroll, and G. Satta (Eds.), *New Developments in Parsing Technology*, pp. 307–321. Kluwer Academic Publishers, Dordrecht, the Netherlands.

- Lavie, A. and M. Tomita (1996). GLR* — an efficient noise-skipping parsing algorithm for context-free grammars. In H. Bunt and M. Tomita (Eds.), *Recent Advances in Parsing Technology*, Chapter 10, pp. 183–200. Kluwer Academic Publishers, Dordrecht, the Netherlands.
- Lee, L. (2002). Fast context-free grammar parsing requires fast Boolean matrix multiplication. *Journal of the ACM* 49(1), 1–15.
- Leiss, H. (1990). On Kilbury's modification of Earley's algorithm. *ACM Transactions on Programming Language and Systems* 12(4), 610–640.
- Ljunglöf, P. (2004). Expressivity and complexity of the grammatical framework. PhD thesis, University of Gothenburg and Chalmers University of Technology, Gothenburg, Sweden.
- Marcus, M. P. (1980). *A Theory of Syntactic Recognition for Natural Language*. MIT Press, Cambridge, MA.
- Mellish, C. S. (1989). Some chart-based techniques for parsing ill-formed input. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics, ACL'89*, Vancouver, Canada, pp. 102–109.
- Menzel, W. (1995). Robust processing of natural language. In *Proceedings of the 19th Annual German Conference on Artificial Intelligence*, Bielefeld, Germany.
- Moore, R. C. (2000). Time as a measure of parsing efficiency. In *Proceedings of the COLING'00 Workshop on Efficiency in Large-Scale Parsing Systems*, Luxembourg.
- Moore, R. C. (2004). Improved left-corner chart parsing for large context-free grammars. In H. Bunt, J. Carroll, and G. Satta (Eds.), *New Developments in Parsing Technology*, pp. 185–201. Kluwer Academic Publishers, Dordrecht, the Netherlands.
- Nakazawa, T. (1991). An extended LR parsing algorithm for grammars using feature-based syntactic categories. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics, EACL'91*, Berlin, Germany.
- Nederhof, M.-J. and J. Sarbo (1996). Increasing the applicability of LR parsing. In H. Bunt and M. Tomita (Eds.), *Recent Advances in Parsing Technology*, pp. 35–58. Kluwer Academic Publishers, Dordrecht, the Netherlands.
- Nederhof, M.-J. and G. Satta (1996). Efficient tabular LR parsing. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics, ACL'96*, Santa Cruz, CA, pp. 239–246.
- Nederhof, M.-J. and G. Satta (2004a). IDL-expressions: A formalism for representing and parsing finite languages in natural language processing. *Journal of Artificial Intelligence Research* 21, 287–317.
- Nederhof, M.-J. and G. Satta (2004b). Tabular parsing. In C. Martin-Vide, V. Mitran, and G. Paun (Eds.), *Formal Languages and Applications*, Volume 148 of *Studies in Fuzziness and Soft Computing*, pp. 529–549. Springer-Verlag, Berlin, Germany.
- Nivre, J. (2002). On statistical methods in natural language processing. In J. Bubenko, Jr. and B. Wangler (Eds.), *Promote IT: Second Conference for the Promotion of Research in IT at New Universities and University Colleges in Sweden*, pp. 684–694, University of Skövde.
- Nivre, J. (2006). *Inductive Dependency Parsing*. Springer-Verlag, New York.
- Nozohoor-Farshi, R. (1991). GLR parsing for ϵ -grammars. In M. Tomita (Ed.), *Generalized LR Parsing*. Kluwer Academic Publishers, Boston, MA.
- Oepen, S. and J. Carroll (2002). Efficient parsing for unification-based grammars. In H. U. D. Flickinger, S. Oepen and J.-I. Tsujii (Eds.), *Collaborative Language Engineering: A Case Study in Efficient Grammar-based Processing*, pp. 195–225. CSLI Publications, Stanford, CA.

- Oepen, S., D. Flickinger, H. Uszkoreit, and J.-I. Tsujii (2000). Introduction to this special issue. *Natural Language Engineering* 6(1), 1–14.
- Pereira, F. C. N. and S. M. Shieber (1984). The semantics of grammar formalisms seen as computer languages. In *Proceedings of the 10th International Conference on Computational Linguistics, COLING'84*, Stanford, CA, pp. 123–129.
- Pereira, F. C. N. and S. M. Shieber (1987). *Prolog and Natural-Language Analysis*, Volume 4 of *CSLI Lecture Notes*. CSLI Publications, Stanford, CA. Reissued in 2002 by Microtome Publishing.

- Pereira, F. C. N. and D. H. D. Warren (1983). Parsing as deduction. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics, ACL'83*, Cambridge, MA, pp. 137–144.
- Pollard, C. and I. Sag (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago, IL.
- Pratt, V. R. (1975). LINGOL – a progress report. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, Georgia, USSR, pp. 422–428.
- Ranta, A. (1994). *Type-Theoretical Grammar*. Oxford University Press, Oxford, U.K.
- Ranta, A. (2004). Grammatical framework, a type-theoretical grammar formalism. *Journal of Functional Programming* 14(2), 145–189.
- Rayner, M., D. Carter, P. Bouillon, V. Digalakis, and M. Wirén (2000). *The Spoken Language Translator*. Cambridge University Press, Cambridge, U.K.
- Rayner, M., B. A. Hockey, and P. Bouillon (2006). *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI Publications, Stanford, CA.
- Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM* 12(1), 23–49.
- Rosé, C. P. and A. Lavie (2001). Balancing robustness and efficiency in unification-augmented context-free parsers for large practical applications. In J.-C. Junqua and G. van Noord (Eds.), *Robustness in Language and Speech Technology*. Kluwer Academic Publishers, Dordrecht, the Netherlands.
- Samuelsson, C. (1994). Notes on LR parser design. In *Proceedings of the 15th International Conference on Computational Linguistics*, Kyoto, Japan, pp. 386–390.
- Samuelsson, C. and M. Rayner (1991). Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, pp. 609–615.
- Sapir, E. (1921). *Language: An Introduction to the Study of Speech*. Harcourt Brace & Co. Orlando, FL.
- Satta, G. (1992). Recognition of linear context-free rewriting systems. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics, ACL'92*, Newark, DE, pp. 89–95.
- Scott, E. and A. Johnstone (2006). Right nullified GLR parsers. *ACM Transactions on Programming Languages and Systems* 28(4), 577–618.
- Scott, E., A. Johnstone, and R. Economopoulos (2007). BRNGLR: A cubic Tomita-style GLR parsing algorithm. *Acta Informatica* 44(6), 427–461.
- Seki, H., T. Matsumara, M. Fujii, and T. Kasami (1991). On multiple context-free grammars. *Theoretical Computer Science* 88, 191–229.
- Shieber, S. M. (1983). Sentence disambiguation by a shift-reduce parsing technique. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics, ACL'83*, Cambridge, MA, pp. 113–118.
- Shieber, S. M. (1985a). Evidence against the context-freeness of natural language. *Linguistics and Philosophy* 8(3), 333–343.
- Shieber, S. M. (1985b). Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics, ACL'85*, Chicago, IL, pp. 145–152.

- Shieber, S. M. (1986). *An Introduction to Unification-based Approaches to Grammar*. Volume 4 of CSLI Lecture Notes. University of Chicago Press, Chicago, IL.
- Shieber, S. M. (1992). *Constraint-Based Grammar Formalisms*. MIT Press, Cambridge, MA.
- Shieber, S. M., Y. Schabes, and F. C. N. Pereira (1995). Principles and implementation of deductive parsing. *Journal of Logic Programming* 24(1–2), 3–36.
- Shieber, S. M., H. Uszkoreit, F. C. N. Pereira, J. J. Robinson, and M. Tyson (1983). The formalism and implementation of PATR-II. In B. J. Grosz and M. E. Stickel (Eds.), *Research on Interactive Acquisition and Use of Knowledge*, Final Report, SRI project number 1894, pp. 39–79. SRI International, Melano Park, CA.

- Sikkel, K. (1998). Parsing schemata and correctness of parsing algorithms. *Theoretical Computer Science* 199, 87–103.
- Sikkel, K. and A. Nijholt (1997). Parsing of context-free languages. In G. Rozenberg and A. Salomaa (Eds.), *The Handbook of Formal Languages*, Volume II, pp. 61–100. Springer-Verlag, Berlin, Germany.
- Slocum, J. (1981). A practical comparison of parsing strategies. In *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics, ACL'81*, Stanford, CA, pp. 1–6.
- Steedman, M. (1985). Dependency and coordination in the grammar of Dutch and English. *Language* 61, 523–568.
- Steedman, M. (1986). Combinators and grammars. In R. Oehrle, E. Bach, and D. Wheeler (Eds.), *Categorial Grammars and Natural Language Structures*, pp. 417–442. Foris Publications, Dordrecht, the Netherlands.
- Steedman, M. J. (1983). Natural and unnatural language processing. In K. Sparck Jones and Y. Wilks (Eds.), *Automatic Natural Language Parsing*, pp. 132–140. Ellis Horwood, Chichester, U.K.
- Tesnière, L. (1959). *Éléments de Syntaxe Structurale*. Librairie C. Klincksieck, Paris, France.
- Thompson, H. S. (1981). Chart parsing and rule schemata in GPSG. In *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics, ACL'81*, Stanford, CA, pp. 167–172.
- Thompson, H. S. (1983). MCHART: A flexible, modular chart parsing system. In *Proceedings of the Third National Conference on Artificial Intelligence*, Washington, DC, pp. 408–410.
- Tomita, M. (1985). *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, Norwell, MA.
- Tomita, M. (1987). An efficient augmented context-free parsing algorithm. *Computational Linguistics* 13(1–2), 31–46.
- Tomita, M. (1988). Graph-structured stack and natural language parsing. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics, ACL'88*, University of New York at Buffalo, Buffalo, NY.
- Toutanova, K., C. D. Manning, S. M. Shieber, D. Flickinger, and S. Oepen (2002). Parse disambiguation for a rich HPSG grammar. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories*, Sozopol, Bulgaria, pp. 253–263.
- Valiant, L. (1975). General context-free recognition in less than cubic time. *Journal of Computer and Systems Sciences* 10(2), 308–315.
- van Noord, G. (1997). An efficient implementation of the head-corner parser. *Computational Linguistics* 23(3), 425–456.
- van Noord, G., G. Bouma, R. Koeling, and M.-J. Nederhof (1999). Robust grammatical analysis for spoken dialogue systems. *Natural Language Engineering* 5(1), 45–93.
- Vijay-Shanker, K. and D. Weir (1993). Parsing some constrained grammar formalisms. *Computational Linguistics* 19(4), 591–636.
- Vijay-Shanker, K. and D. Weir (1994). The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory* 27(6), 511–546.
- Vijay-Shanker, K., D. Weir, and A. K. Joshi (1987). Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics, ACL'87*, Stanford, CA.

- Ward, W. (1989). Understanding spontaneous speech. In *Proceedings of the Workshop on Speech and Natural Language, HLT '89*, Philadelphia, PA, pp. 137–141.
- Wirén, M. (1987). A comparison of rule-invocation strategies in context-free chart parsing. In *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics EACL'87*, Copenhagen, Denmark.
- Woods, W. A. (1970). Transition network grammars for natural language analysis. *Communications of the ACM* 13(10), 591–606.
- Woods, W. A. (1973). An experimental parsing system for transition network grammars. In R. Rustin (Ed.), *Natural Language Processing*, pp. 111–154. Algorithmics Press, New York.

- Woods, W. A., R. M. Kaplan, and B. Nash-Webber (1972). The lunar sciences natural language information system: final report. BBN Report 2378, Bolt, Beranek, and Newman, Inc., Cambridge, MA.
- Yngve, V. H. (1955). Syntax and the problem of multiple meaning. In W. N. Locke and A. D. Booth (Eds.), *Machine Translation of Languages*, pp. 208–226. MIT Press, Cambridge, MA.
- Younger, D. H. (1967). Recognition of context-free languages in time n^3 . *Information and Control* 10(2), 189–208.

5

Semantic Analysis

5.1	Basic Concepts and Issues in Natural Language Semantics ...	94
5.2	Theories and Approaches to Semantic Representation.....	95
	Logical Approaches · Discourse Representation Theory · Pustejovsky's Generative Lexicon · Natural Semantic Metalanguage · Object-Oriented Semantics	
5.3	Relational Issues in Lexical Semantics	103
	Sense Relations and Ontologies · Roles	
5.4	Fine-Grained Lexical-Semantic Analysis: Three Case Studies .	107
	Emotional Meanings: "Sadness" and "Worry" in English and Chinese · Ethnogeographical Categories: "Rivers" and "Creeks" · Functional Macro- Categories	
5.5	Prospectus and "Hard Problems"	113
	Acknowledgments	114
	References	114

Cliff Goddard
University of New England

Andrea C. Schalley
Griffith University

A classic NLP interpretation of semantic analysis was provided by Poesio (2000) in the first edition of the *Handbook of Natural Language Processing*:

The ultimate goal, for humans as well as natural language-processing (NLP) systems, is to understand the utterance—which, depending on the circumstances, may mean incorporating information provided by the utterance into one's own knowledge base or, more in general performing some action in response to it. 'Understanding' an utterance is a complex process, that depends on the results of parsing, as well as on lexical information, context, and commonsense reasoning... (Poesio 2000: 93).

For extended texts, specific NLP applications of semantic analysis may include information retrieval, information extraction, text summarization, data-mining, and machine translation and translation aids. Semantic analysis is also pertinent for much shorter texts, right down to the single word level, for example, in understanding user queries and matching user requirements to available data. Semantic analysis is also of high relevance in efforts to improve Web ontologies and knowledge representation systems.

Two important themes form the grounding for the discussion in this chapter. First, there is great value in conducting semantic analysis, as far as possible, in such a way as to reflect the cognitive reality of ordinary speakers. This makes it easier to model the intuitions of native speakers and to simulate their inferencing processes, and it facilitates human–computer interactions via querying processes, and the like. Second, there is concern over to what extent it will be possible to achieve comparability, and, more ambitiously, interoperability, between different systems of semantic description. For both reasons, it is

highly desirable if semantic analyses can be conducted in terms of intuitive representations, be it in simple ordinary language or by way of other intuitively accessible representations.

5.1 Basic Concepts and Issues in Natural Language Semantics

In general linguistics, semantic analysis refers to analyzing the meanings of words, fixed expressions, whole sentences, and utterances in context. In practice, this means translating original expressions into some kind of semantic metalanguage. The major theoretical issues in semantic analysis therefore turn on the nature of the metalanguage or equivalent representational system (see Section 5.2). Many approaches under the influence of philosophical logic have restricted themselves to truth-conditional meaning, but such analyses are too narrow to enable a comprehensive account of ordinary language use or to enable many practically required applications, especially those involving human–computer interfacing or naïve reasoning by ordinary users. Unfortunately, there is even less consensus in the field of linguistic semantics than in other subfields of linguistics, such as syntax, morphology, and phonology. NLP practitioners interested in semantic analysis nevertheless need to become familiar with standard concepts and procedures in semantics and lexicology. The following is a tutorial introduction. It will provide the reader with foundational knowledge on linguistic semantics. It is not intended to give an overview of applications within computational linguistics or to introduce hands-on methods, but rather aims to provide basic theoretical background and references necessary for further study, as well as three case studies.

There is a traditional division made between lexical semantics, which concerns itself with the meanings of words and fixed word combinations, and supralexical (combinational, or compositional) semantics, which concerns itself with the meanings of the indefinitely large number of word combinations—phrases and sentences—allowable under the grammar. While there is some obvious appeal and validity to this division, it is increasingly recognized that word-level semantics and grammatical semantics interact and interpenetrate in various ways. Many linguists now prefer to speak of lexicogrammar, rather than to maintain a strict lexicon-grammar distinction. In part, this is because it is evident that the combinatorial potential of words is largely determined by their meanings, in part because it is clear that many grammatical constructions have construction-specific meanings; for example, the construction *to have a VP* (*to have a drink, a swim*, etc.) has meaning components additional to those belonging to the words involved (Wierzbicka 1982; Goldberg 1995; Goddard 2000; Fried and Östman 2004). Despite the artificiality of rigidly separating lexical semantics from other domains of semantic analysis, lexical semantics remains the locus of many of the hard problems, especially in crosslinguistic contexts. Partly this is because lexical semantics has received relatively little attention in syntax-driven models of language or in formal (logic-based) semantics.

It is widely recognized that the overriding problems in semantic analysis are how to avoid circularity and how to avoid infinite regress. Most approaches concur that the solution is to ground the analysis in a terminal set of primitive elements, but they differ on the nature of the primitives (are they elements of natural language or creations of the analyst? are they of a structural-procedural nature or more encompassing than this? are they language-specific or universal?). Approaches also differ on the extent to which they envisage that semantic analysis can be precise and exhaustive (how fine-grained can one expect a semantic analysis to be? are semantic analyses expected to be complete or can they be underspecified? if the latter, how exactly are the missing details to be filled in?).

A major divide in semantic theory turns on the question of whether it is possible to draw a strict line between semantic content, in the sense of content encoded in the lexicogrammar, and general encyclopedic knowledge. Whatever one's position on this issue, it is universally acknowledged that ordinary language use involves a more or less seamless integration of linguistic knowledge, cultural conventions, and real-world knowledge.

In general terms, the primary evidence for linguistic semantics comes from native speaker interpretations of the use of linguistic expressions in context (including their entailments and implications), from naturalistic observation of language in use, and from the distribution of linguistic expressions, that is,

patterns of usage, collocation, and frequency, discoverable using the techniques of corpus linguistics (see Chapter 7).

One frequently identified requirement for semantic analysis in NLP goes under the heading of ambiguity resolution. From a machine point of view, many human utterances are open to multiple interpretations, because words may have more than one meaning (lexical ambiguity), or because certain words, such as quantifiers, modals, or negative operators may apply to different stretches of text (scopal ambiguity), or because the intended reference of pronouns or other referring expressions may be unclear (referential ambiguity).

In relation to lexical ambiguities, it is usual to distinguish between homonymy (different words with the same form, either in sound or writing, for example, *light* (vs. *dark*) and *light* (vs. *heavy*), *son* and *sun*, and polysemy (different senses of the same word, for example, the several senses of the words *hot* and *see*). Both phenomena are problematical for NLP, but polysemy poses greater problems, because the meaning differences concerned, and the associated syntactic and other formal differences, are typically more subtle. Mishandling of polysemy is a common failing of semantic analysis: both the positing of false polysemy and failure to recognize real polysemy (Wierzbicka 1996: Chap. 9; Goddard 2000). The former problem is very common in conventional dictionaries, including Collins Cobuild and Longman, and also in WordNet. The latter is more common in theoretical semantics, where theorists are often reluctant to face up to the complexities of lexical meanings. Further problems for lexical semantics are posed by the widespread existence of figurative expressions and/or multi-word units (fixed expressions such as *by and large*, *be carried away*, or *kick the bucket*), whose meanings are not predictable from the meanings of the individual words taken separately.

5.2 Theories and Approaches to Semantic Representation

Various theories and approaches to semantic representation can be roughly ranged along two dimensions: (1) formal vs. cognitive and (2) compositional vs. lexical. Formal theories have been strongly advocated since the late 1960s (e.g., Montague 1973, 1974; Cann 1994; Lappin 1997; Portner and Partee 2002; Gutiérrez-Rexach 2003), while cognitive approaches have become popular in the last three decades (e.g., Fauconnier 1985; Johnson 1987; Lakoff 1987; Langacker 1987, 1990, 1991; Jackendoff 1990, 2002; Wierzbicka 1988, 1992, 1996; Talmy 2000; Geeraerts 2002; Croft and Cruse 2003; Cruse 2004), driven also by influences from cognitive science and psychology. Compositional semantics is concerned with the bottom-up construction of meaning, starting with the lexical items, whose meanings are generally treated as given, that is, are left unanalyzed. Lexical semantics, on the other hand, aims at precisely analyzing the meanings of lexical items, either by analyzing their internal structure and content (decompositional approaches) or by representing their relations to other elements in the lexicon (relational approaches, see Section 5.3).

This section surveys some of the theories and approaches, though due to limitations of space this can only be done in a cursory fashion. Several approaches will have to remain unmentioned here, but the interested reader is referred to the accompanying wiki for an expanded reading list. We will start with a formal-compositional approach and move toward more cognitive-lexical approaches.

5.2.1 Logical Approaches

Logical approaches to meaning generally address problems in compositionality, on the assumption (the so-called *principle of compositionality*, attributed to Frege) that the meanings of supralexical expressions are determined by the meanings of their parts and the way in which those parts are combined.

There is no universal logic that covers all aspects of linguistic meaning and characterizes all valid arguments or relationships between the meanings of linguistic expressions (Gamut 1991 Vol. I: 7). Different logical systems have been and are being developed for linguistic semantics and NLP. The most well known and widespread is predicate logic, in which properties of sets of objects can be expressed via predicates, logical connectives, and quantifiers. This is done by providing a “syntax” (i.e., a specification

how the elements of the logical language can be combined to form well-formed logical expressions) and a

“semantics” (an interpretation of the logical expressions, a specification of what these expressions mean within the logical system). Examples of predicate logic representations are given in (1b) and (2b), which represent the semantic interpretation or meaning of the sentences in (1a) and (2a), respectively. In these formulae, x is a ‘variable,’ k a ‘term’ (denoting a particular object or entity), **politician**, **mortal**, **like**, etc. are predicates (of different arity), $\wedge$, $\rightarrow$ are ‘connectives,’ and $\exists$, $\forall$ are the existential quantifier and universal quantifier, respectively. Negation can also be expressed in predicate logic, using the symbol $\neg$ or a variant.

- (1) a. *Some politicians are mortal.*
 b. $\exists x (\text{politician}(x) \wedge \text{mortal}(x))$
 [There is an x (at least one) so that x is a politician and x is mortal.]
- (2) a. *All Australian students like Kevin Rudd.*
 b. $\forall x ((\text{student}(x) \wedge \text{Australian}(x)) \rightarrow \text{like}(x, k))$
 [For all x with x being a student and Australian, x likes Kevin Rudd.]

Notice that, as mentioned, there is no analysis of the meanings of the predicates, which correspond to the lexical items in the original sentences, for example, *politician*, *mortal*, *student*, etc. Notice also the “constructed” and somewhat artificial sounding character of the example sentences concerned, which is typical of much work in the logical tradition.

Predicate logic also includes a specification of valid conclusions or inferences that can be drawn: a proof theory comprises inference rules whose operation determines which sentences must be true given that some other sentences are true (Poesio 2000). The best known example of such an inference rule is the rule of *modus ponens*: If P is the case and $P \rightarrow Q$ is the case, then Q is the case (cf. (3)):

- (3) a. Modus ponens:
 (i) P (premise)
 (ii) $P \rightarrow Q$ (premise)
 (iii) Q (conclusion)
 b. (i) *Conrad is tired* (P : **tired**(c))
 (ii) *Whenever Conrad is tired, he sleeps* (P : **tired**(c), Q : **sleep**(c), $P \rightarrow Q$)
 (iii) *Conrad sleeps* (Q : **sleep**(c))

In the interpretation of sentences in formal semantics, the meaning of a sentence is often equated with its truth conditions, that is, the conditions under which the sentence is true. This has led to an application of model theory (Dowty et al. 1981) to natural language semantics. The logical language is interpreted in a way that for the logical statements general truth conditions are formulated, which result in concrete truth values under concrete models (or possible worlds). An alternative approach to truth-conditional and possible world semantics is situation semantics (Barwise and Perry 1983), in which situations rather than truth values are assigned to sentences as referents.

Although sometimes presented as a general-purpose theory of knowledge, predicate logic is not powerful enough to represent the intricacies of semantic meaning and is fundamentally different from human reasoning (Poesio 2000). It has nevertheless found application in logic programming, which in turn has been successfully applied in linguistic semantics (e.g., Lambalgen and Hamm 2005). For detailed introductions to logic formalisms, including lambda calculus and typed logical approaches,* see Gamut (1991) and Blackburn and Bos (2005). Amongst other things, lambda calculus provides a way of converting open formulae (those containing free variables) into complex one-place predicates to allow their use as predicates in other formulae. For instance, in **student**(x) $\wedge$ **Australian**(x) the variable x is

* Types are assigned to expression parts, allowing the computation of the overall expression’s type. This allows the well-formedness of a sentence to be checked. If α is an expression of type $\langle m, n \rangle$, and β is an expression of type m , then the application of α to β , $\alpha(\beta)$, will have the type n . In linguistic semantics, variables and terms are generally assigned the type e (‘entity’), and formulae

the type t ('truth value'). Then one-place predicates have the type $\langle e, t \rangle$: The application of the one-place predicate **sleep** to the term c (*Conrad*) yields the type formula **sleep**(c).

not bound. The lambda operator λ converts this open formula into a complex one-place predicate: λx (**student**(x) $\wedge$ **Australian**(x)), which is read as “those x for which it is the case that they are a student and Australian.”

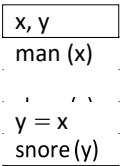
5.2.2 Discourse Representation Theory

Discourse representation theory (DRT) was developed in the early 1980s by Kamp (1981) (cf. Kamp and Reyle 1993; Blackburn and Bos 1999; van Eijck 2006; Geurts and Beaver 2008) in order to capture the semantics of discourses or texts, that is, coherent sequences of sentences or utterances, as opposed to isolated sentences or utterances. The basic idea is that as a discourse or text unfolds the hearer builds up a mental representation (represented by a so-called discourse representation structure, DRS), and that every incoming sentence prompts additions to this representation. It is thus a dynamic approach to natural language semantics (as it is in the similar, independently developed File Change Semantics (Heim 1982, 1983)).

DRT formally requires the following components (Geurts and Beaver 2008): (1) a formal definition of the representation language, consisting of (a) a recursive definition of the set of all well-formed DRSs, and (b) a model-theoretic semantics for the members of this set; and (2) a construction procedure specifying how a DRS is to be extended when new information becomes available.

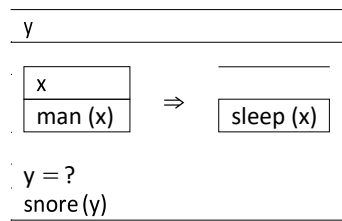
ADRS consists of a universe of so-called discourse referents (these represent the objects under discussion in the discourse), and conditions applying to these discourse referents (these encode the information that has been accumulated on the discourse referents and are given in first-order predicate logic). A simple example is given in (4). As (4) shows, a DRS is presented in a graphical format, as a rectangle with two compartments. The discourse referents are listed in the upper compartment and the conditions are given in the lower compartment. The two discourse referents in the example (x and y) denote *a man* and *he*, respectively. In the example, *a man* and *he* are anaphorically linked through the condition $y = x$, that is, the pronoun *he* refers back to *a man*. The linking itself is achieved as part of the construction procedure referred to above.

(4) *A man sleeps. He snores.*



Recursiveness is an important feature. DRSs can comprise conditions that contain other DRSs. An example is given in (5). Notice that according to native speaker intuition this sequence is anomalous: though on the face of it *every man* is a singular noun-phrase, the pronoun *he* cannot refer back to it.

(5) *Every man sleeps. He snores.*



In the DRT representation, the quantification in the first sentence of (5) results in an if-then condition: if x is a man, then x sleeps. This condition is expressed through a conditional ($A \Rightarrow B$) involving two

DRSs. This results in x being declared at a lower level than y , namely, in the nested DRS that is part of the

conditional, which means that *x* is not an accessible discourse referent for *y*, and hence that *every man* cannot be an antecedent for *he*, in correspondence with native speaker intuition.

The DRT approach is well suited to dealing with indefinite noun phrases (and the question of when to introduce a new discourse referent, cf. also Karttunen 1976), presupposition, quantification, tense, and anaphora resolution. Discourse Representation Theory is thus seen as having “enabled perspicuous treatments of a range of natural language phenomena that have proved recalcitrant over many years” (Geurts and Beaver 2008) to formal approaches. In addition, inference systems have been developed (Saurer 1993; Kamp and Reyle 1996) and implementations employing Prolog (Blackburn and Bos 1999, 2005). Extensions of DRT have also been developed. For the purposes of NLP, the most relevant is Segmented Discourse Representation Theory (SDRT; Asher 1993; Asher and Lascarides 2003). It combines the insights of DRT and dynamic semantics on anaphora with a theory of discourse structure in which each clause plays one or more rhetorical functions within the discourse and entertains rhetorical relations to other clauses, such as “explanation,” “elaboration,” “narration,” and “contrast.”

5.2.3 Pustejovsky’s Generative Lexicon

Another dynamic view of semantics, but focusing on lexical items, is Pustejovsky’s (1991a,b, 1995, 2001) Generative Lexicon theory. He states: “our aim is to provide an adequate description of how our language expressions have content, and how this content appears to undergo continuous modification and modulation in new contexts” (Pustejovsky 2001: 52).

Pustejovsky posits that within particular contexts, lexical items assume different senses. For example, the adjective *good* is understood differently in the following four contexts: (a) *a good umbrella* (an umbrella that guards well against rain), (b) *a good meal* (a meal that is delicious or nourishing), (c) *a good teacher* (a teacher who educates well), (d) *a good movie* (a movie that is entertaining or thought provoking). He develops “the idea of a lexicon in which senses [of words/lexical items, CG/AS] in context can be flexibly derived on the basis of a rich multilevel representation and generative devices” (Behrens 1998: 108). This lexicon is characterized as a computational system, with the multilevel representation involving at least the following four levels (Pustejovsky 1995: 61):

1. *Argument structure*: Specification of number and type of logical arguments and how they are realized syntactically.
2. *Event structure*: Definition of the event type of a lexical item and a phrase. The event type sorts include states, processes, and transitions; sub-event structuring is possible.
3. *Qualia structure*: Modes of explanation, comprising qualia (singular: quale) of four kinds: *constitutive* (what an object is made of), *formal* (what an object is—that which distinguishes it within a larger domain), *telic* (what the purpose or function of an object is), and *agentive* (how the object came into being, factors involved in its origin or coming about).
4. *Lexical Inheritance Structure*: Identification of how a lexical structure is related to other structures in the lexicon and its contribution to the global organization of a lexicon.

The multilevel representation is given in a structure similar to HPSG structures (Head Driven Phrase Structure Grammar; Pollard and Sag 1994). An example of the lexical representation for the English verb *build* is given in Figure 5.1 (Pustejovsky 1995: 82).

The event structure shows that *build* is analyzed as involving a process (e_1) followed by a resultant state (e_2) and ordered by the relation “exhaustive ordered part of ($<_e$).” The head of the event structure, and hence foregrounded, is the process e_1 . The argument structure lists three arguments—two true arguments (ARG_1 , ARG_2) that are syntactically realized parameters of the lexical item *build* (*John built a house*), and one default argument ($D-ARG_1$), a parameter that participates in the expression of the qualia but is not necessarily expressed syntactically (*John built a house out of bricks*). For the arguments, their characteristics are being noted in that their ontological restriction and qualia are listed: ARG_1 , for instance,

is restricted to arguments that are animate individuals and of a physical nature, while ARG2 is—also a physical object—an artifact and made out of D-ARG1 (as the cross-reference via the boxed number ‘3’

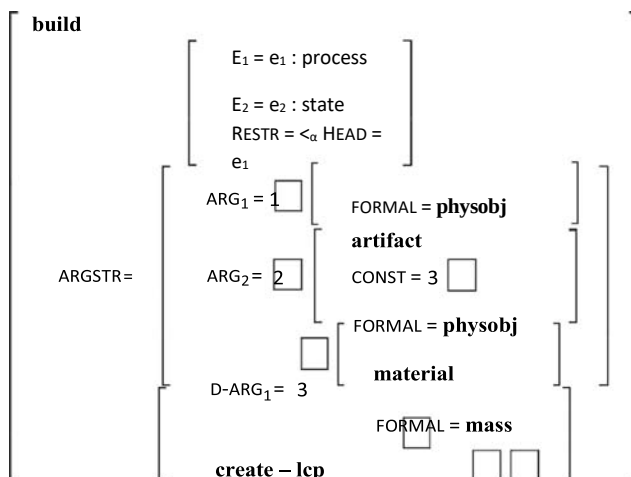


FIGURE 5.1 The lexical representation for the English verb *build*. (From Pustejovsky, J., *The Generative Lexicon*, MIT Press, Cambridge, MA, 1995, 92. With permission.)

indicates). The qualia structure outlines that *build* is overall a **create** event (event or similar entity, cf. Zaefferer 2002), and that what is created comes about by a **build_act**—a process (e_1) performed by ARG₁ ('1') and using D-ARG₁ ('3'). The result is the **exist** resultant state (e_2) of ARG₂. Notice that expressions such as **create**, **build_act**, **physobj** (physical object), etc., are taken as givens, that is, they are not analyzed, although they are embedded in inheritance structures.

The generative devices in Pustejovsky's system connect the levels of representation with the aim of providing the compositional interpretation of the lexical elements in context. They include those listed below (Pustejovsky 1995: 61–62). For further details and a formalization of the generative devices, see Pustejovsky (1995, esp. Chap. 7).

1. *Type coercion*: A lexically governed type shifting; a semantic operation that converts an argument to the type which is expected by a function, where it would otherwise result in a type error (Pustejovsky 1995: 111). For example, if *want* is assumed to have a proposition as an argument, the non-propositional argument *a beer* in *Mary wants a beer* is coerced into a propositional type (on the basis of the telic quale of *beer drinking*); hence, *Mary wants a beer* amounts to 'Mary wants to drink a beer.'
2. *Selective binding*: A lexical item operates specifically on the substructure of a phrase, without changing the overall type in the composition—an adjective can modify a particular aspect of noun qualia structure it is in composition with. For example, *a good knife* describes a knife in which the cutting is good (i.e., which cuts well): the adjective *good* functions as an event predicate and is able to selectively modify the event description in the telic quale of the noun (which is cutting).
3. *Co-composition*: Multiple elements within a phrase behave as functors, generating new non-lexicalized senses for the words in composition—the argument qualia structure can modify the semantic type of the predicate. For example, *bake* is generally analyzed as having a change of state meaning (as in *bake potatoes*), but in *bake a cake*, the meaning of the predicate *bake* is modified into a creation meaning. This is because the agentive quale of the cake "makes reference to the very process within which it is embedded in this phrase" (Pustejovsky 1995: 123), namely, the baking.

The Generative Lexicon has a different focus from the logic-inspired approaches to semantics. It does not

aim principally at capturing or explaining phenomena such as quantification, anaphora, or presupposition. It is geared towards a detailed, compositional lexical approach to linguistic semantics, while at the same time providing devices that allow for the computing of compositional meaning in context.

5.2.4 Natural Semantic Metalanguage

Natural semantic metalanguage (NSM) is a decompositional system based on empirically established semantic primes, that is, simple indefinable meanings which appear to be present as word-meanings in all languages (Wierzbicka 1996; Goddard 1998; Goddard and Wierzbicka 2002; Peeters 2006; Goddard 2008). The NSM system uses a metalanguage, which is essentially a standardized subset of natural language: a small subset of word-meanings (63 in number, see Table 5.1), together with a subset of their associated syntactic (combinatorial) properties. Table 5.1 is presented in its English version, but comparable tables of semantic primes have been drawn up for many languages, including Russian, French, Spanish, Chinese, Japanese, Korean, and Malay. Semantic primes and their grammar are claimed to represent a kind of “intersection” of all languages.

NSM is a cognitive theory. Its adherents argue that simple words of ordinary language provide a better representational medium for ordinary cognition than the more technical metalanguages used by other cognitive theories, such as Jackendoff’s (1990, 2002) Conceptual Semantics or Wunderlich’s (1996, 1997) Lexical Decomposition Grammar. They also argue that any system of meaning representation is necessarily grounded in ordinary language, whether its proponents recognize it or not, because as Lyons (1977: 12) put it: “any formalism is parasitic upon the ordinary everyday use of language, in that it must be understood intuitively on the basis of ordinary language.” In relation to crosslinguistic semantics, NSM theorists charge that other cognitivist approaches typically incur “terminological ethnocentrism” (Goddard 2002, 2006; Wierzbicka 2009a), that is, the imposition of inauthentic nonnative categories, because they usually treat the language-specific categories of English as if they represent objective language-independent categories.

The formal mode of meaning representation in the NSM approach is the semantic explication. This is a reductive paraphrase, that is, an attempt to say in other words what a speaker is saying when he or she utters the expression being explicated. Unlike other decompositional systems, reductive paraphrase attempts to capture an insider perspective—with its sometimes naïve first-person quality, rather than the sophisticated outsider perspective of an expert linguist, logician, etc. Originating with Wierzbicka (1972),

TABLE 5.1 Semantic Primes, Grouped into Related Categories

I, YOU, SOMEONE, SOMETHING/THING, PEOPLE, BODY	Substantives
KIND, PART	Relational substantives
THIS, THESAME, OTHER/ELSE	Determiners
ONE, TWO, SOME, ALL, MUCH/MANY	Quantifiers
GOOD, BAD	Evaluators
BIG, SMALL	Descriptors
KNOW, THINK, WANT, FEEL, SEE, HEAR	Mental predicates
SAY, WORDS, TRUE	Speech
DO, HAPPEN, MOVE, TOUCH	Actions, events, movement, contact
BE (SOMEWHERE), THERE IS, HAVE, BE (SOMEONE/SOMETHING)	Location, existence, possession, specification
LIVE, DIE	Life and death

WHEN/TIME, NOW, BEFORE, AFTER, A LONG TIME, A SHORT TIME, FOR SOME TIME, MOMENT	Time
WHERE/PLACE, HERE, ABOVE, BELOW, FAR, NEAR, SIDE, INSIDE	Space
NOT, MAYBE, CAN,BECAUSE, IF	Logical concepts
VERY, MORE	Intensifier, augmentor
LIKE/WAY	Similarity

Notes: Primes exist as the meanings of lexical units (not at the level of lexemes). Exponents of primes may be words, bound morphemes, or phrasemes. They can be formally complex. They can have combinatorial variants (allolexes). Each prime has well-specified syntactic (combinatorial) properties.

the NSM system has been developed and refined over some 35 years. There is a large body of descriptive empirical work in the framework, with hundreds of published explications. Some lexicon areas that have been explored in great depth are emotions and other mental states, speech-acts, causatives, cultural values, natural kind words, concrete objects, physical activity verbs, and discourse particles. The approach also has been extensively applied to grammatical semantics and pragmatics, but it is not possible to cover these aspects here. Though the NSM approach is arguably the best developed theory of lexical semantics on the contemporary scene, it so far has had minimal application to NLP. (Exceptions include Andrews (2006) on semantic composition for NSM using glue-logic, cf. Dalrymple (2001: 217–254); and Francesco Zamblera's ongoing work on a PROLOG-based parser-generator for NSM. For more information, refer to this chapter's section in the accompanying wiki.)

A simple example of an NSM explication is given in [A] for the English verb *to break*, in one of its several senses. The explication applies only to the sense of the word found in examples like *to break a stick* (an egg, a lightbulb, a vase, a model plane). Its successive components indicate action, concurrent effect, aspect, and a result ('this thing was not one thing anymore'); the final "subjective" component indicates that the result is seen as irreversible. Interestingly, many languages—including Chinese, Malay, and Lao—lack a comparably broad verb that subsumes many different manners of "breaking" (Majid and Bowerman 2007).

[A] Semantic explication for *Someone X broke something Y*

- a. someone X did something to something Y
- b. because of this, something happened to this something Y at the same time
- c. it happened in one moment
- d. because of this, afterwards this thing was not one thing anymore
- e. people can think about it like this: "it can't be one thing anymore"

NSM researchers recognize that for many words in the concrete vocabulary, it is not possible to produce plausible explications directly in terms of semantic primes alone (Wierzbicka 1991; Goddard 2007, 2008). Rather, the explications typically require a combination of semantic primes and certain complex lexical meanings known in NSM theory as "semantic molecules." Though ultimately decomposable into combinations of primes, semantic molecules function as building blocks in the structure of other, yet more complex concepts. For example, explications for *sparrow* and *eagle* must include 'bird [M]' as a semantic molecule; explications for *walk* and *run* must include 'feet [M]' and 'ground [M]'. (In NSM explications, semantic molecules are marked as such by the notation [M].) The concept of semantic molecules is similar to that of intermediate-level concepts in the semantic practice of the Moscow School (Apresjan 1992, 2000; Wanner 1996, 2007), but with the constraint that they must be meanings of lexical units in the language concerned. Semantic molecules can be nested, one within the other, creating chains of semantic dependency. Up to four levels of nesting are attested. Some semantic molecules appear to be universal or near-universal, but many are clearly language- and culture-specific. Semantic molecules vary in their degree of productivity and in how widely they range across the lexicon.

NSM researchers estimate there to be about 150–250 productive semantic molecules in English. They are drawn from at least the following categories (examples given are non-exhaustive): (a) parts of the body: 'hands,' 'mouth,' 'legs'; (b) physical descriptors: 'long,' 'round,' 'flat,' 'hard,' 'sharp,' 'straight'; (c) activities and actions: 'eat,' 'drink,' 'sit,' 'kill,' 'pick up,' 'fight with'; (d) expressive/communicative actions: 'laugh,' 'sing,' 'write,' 'read'; (e) topological terms: 'edges,' 'hole'; (f) life-form words: 'creature,' 'animal,' 'bird,' 'fish,' 'tree'; (g) environment: 'ground,' 'sky,' 'sun,' 'water,' 'fire,' 'day,' 'night'; (h) materials: 'wood,' 'stone,' 'metal,' 'glass,' 'paper'; (i) mechanical parts and technology: 'wheel,' 'pipe,' 'wire,' 'engine,' 'electricity,' 'machine'; (j) transport: 'car,' 'train,' 'boat,' 'plane'; (k) social categories and kin roles: 'men,' 'women,' 'children,' 'mother,' 'father'; (l) important cultural concepts: 'money,' 'book,' 'color,' 'number,' 'God.' All of these posited semantic molecules, it must be emphasized, can ultimately be explicated,

without circularity, into the fundamental underlying metalanguage of semantic primes.

The concept of semantic molecules leads to new ways of understanding semantic complexity in the lexicon, quite different to anything envisaged in other cognitive or generative approaches to lexical semantics, and very different to structuralist and logic-based approaches. Many lexical semantic structures appear to have a kind of “gangly and lumpy” quality — lengthy strings of simple semantic primes interspersed with semantically dense molecules (for examples, see Section 5.4). Semantic molecules enable a great compression of semantic complexity, but at the same time this complexity is disguised by its being encapsulated and telescoped into lexical units embedded one in the other, like a set of Russian dolls.

5.2.5 Object-Oriented Semantics

Object-oriented semantics is a new field in linguistic semantics. Although it is rather restricted in its semantic application domains so far — mainly applied to the representation of verbal meaning — it promises to become relevant for NLP applications in the future, and due to the large body of research in computer science a wealth of resources is already available for object-oriented systems in general. There is some overlap with Pustejovsky’s ideas, as he himself observes:

When we combine the qualia structure of a NP with the argument structure of a verb, we begin to see a richer notion of compositionality emerging, one that looks very much like object-oriented approaches to programming. (Pustejovsky 1991a: 427)

The basic motive behind deploying the computational object-oriented paradigm to linguistic semantics is, however, its intuitive accessibility. The human cognitive system centers around entities and what they are like, how they are related to each other, what happens to them and what they do, and how they interact with one another. This corresponds to the object-oriented approach, in which the concept of “object” is central, whose characteristics, relations to other entities, behavior, and interactions are modeled in a rigorous way. This correspondence between object-orientation and cognitive organization strongly suggests the application of object-orientation for the representational task at hand.

Schalley (2004a,b) introduces a decompositional representational framework for verbal semantics, the Unified Entity Representation (UER). It is based on the Unified Modeling Language (UML; cf. Object Management Group 1997–2009), the standard formalism for object-oriented software design and analysis. The UER adopts the graphical nature of the UML as well as the UML’s general architecture. UER diagrams are composed of well-defined graphical modeling elements that represent conceptual categories. Conceptual containments, attachments, and relations are thus expressed by graphical entailments, attachments, and relations.

The lexical representation is much more detailed than in the Generative Lexicon, and the graphical modeling elements highlight the internal structure of verbal meaning. An example is given in Figure 5.2 (Schalley 2004b: 787).

The modeling of the transitive *wake up* (as in *Mary woke up her husband*), dubbed **wake_up_2** in Figure 5.2, contains a detailed modeling of the “static structure” of verbal meaning (primarily of the participants in the wake up eventivity, their characteristics, and roles) and of the “dynamic structure” (what is happening to the participants in the wake up eventivity? which states and transitions do the participants undergo? how do they interact?). Roughly, there is an instigating actor *x* (either a volitional agent or an non-volitional effector) who/which does something that causes (triggers) a change of state in the animate patient undergoer *y*, so that *y* is in the passive state of being awake as a result. A detailed interpretation of the diagrammatic representation, in particular a definition of the diagrammatic modeling elements, can be found in Schalley (2004a).

Work is underway on supralexical semantics and the development of an object-oriented discourse representation system. The ability to model verbal meanings in fine detail is an important prerequisite for such an enterprise, as verbs are the central elements of supralexical expressions. The eventivity structure

underlying verbal meaning constitutes a scaffolding for utterance interpretation, which is instantiated

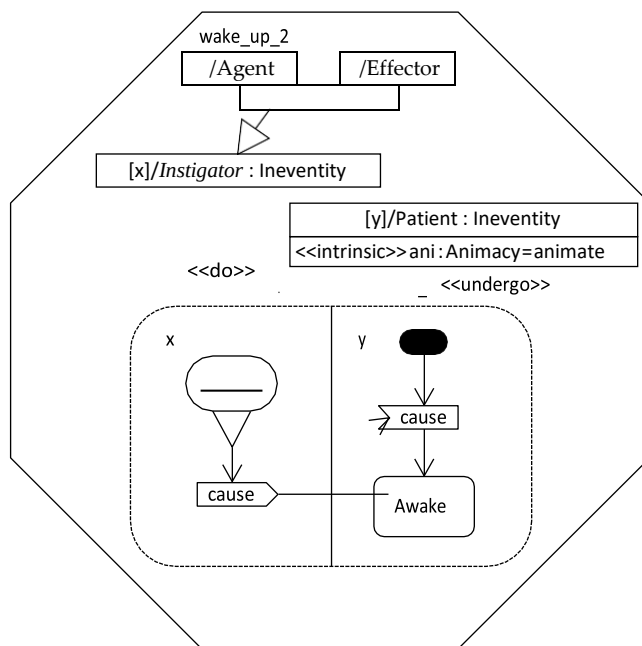


FIGURE 5.2 UER diagrammatic modeling for transitive verb *wake up*. (Schalley 2004b: 787, cf. Schalley 2004a: 76)

and contextually completed in the situation. The type-instance dichotomy of object-oriented systems is expected to reflect this perfectly. While the modeling of lexical meaning is a task for the type level, the representation of utterance meaning is a task for the instance or user objects layer.

5.3 Relational Issues in Lexical Semantics

Linguistic expressions are not isolated, but are interrelated in many ways. In this section, we focus on relational issues in lexical semantics.

5.3.1 Sense Relations and Ontologies

Sense relations—semantic relations between lexical elements—form the basis for word nets, such as the electronic lexical database WordNet (Fellbaum 1998a,b) and similar approaches for languages other than English, for example, the multilingual lexical database EuroWordNet (Vossen 1998, 2001). The large number of NLP applications drawing on WordNet—in areas such as information retrieval and extraction (including audio and video retrieval and the improvement of internet search technology), disambiguation, document structuring and categorization (cf., e.g., Fellbaum 1998b; Morato et al. 2004; Huang et al. in press), to mention a few—show that there is a high demand for information on lexical relations and linguistically informed ontologies. Although a rather new field, the interface between ontologies and

linguistic structure is becoming a vibrant and influential area in knowledge representation and NLP (cf. Chapter 24; Schalley and Zaefferer 2007), also due to increasingly popular research on Semantic Web and

similar areas.* In the following, we will introduce a few relevant theoretical notions. For more information on resources and tools, see Huang et al. (in press).

Sense relations can be seen as revelatory of the semantic structure of the lexicon. There are both horizontal and vertical sense relations. Horizontal relations include *synonymy* (sameness of meaning of different linguistic forms, such as *Orange* and *Apfelsine* in German—both meaning ‘orange’) and various relations that can be subsumed under the general notion of *opposition* (Cruse 1986). These include incompatibility (mutually exclusive in a particular context, e.g., *fin*—*foot*), antonymy (gradable, directionally opposed, e.g., *big*—*small*), complementarity (exhaustive division of conceptual domain into mutually exclusive compartments, e.g., *aunt*—*uncle*, *possible*—*impossible*), conversity (static directional opposites: specification of one lexical element in relation to another along some axis, for example, *above*—*below*), and reversivity (dynamic directional opposites: motion or change in opposite ways, e.g., *ascend*—*descend*).

The two principal vertical relations are hyponymy and meronymy. Hyponymy is when the meaning of one lexical element, the hyponym, is more specific than the meaning of the other, the hypernym (e.g., *dog*—*animal*). It is often assumed that hyponymy corresponds to a taxonomic (kind of) relationship, but this assumption is incorrect (see Section 5.4.3). Lexical items that are hyponyms of the same lexical element and belong to the same level in the structure are called co-hyponyms (*dog*, *cat*, *horse* are co-hyponyms of *animal*). Meronymy is when the meaning of one lexical element specifies that its referent is ‘part of’ the referent of another lexical element, for example, *hand*—*body* (for more on meronymy, refer to this chapter’s section in the accompanying wiki). WordNet uses another vertical sense relation for the verbal lexicon: troponymy—where one verb specifies a certain manner of carrying out the action referred to by the other verb, such as in *amble*—*walk* or *shelve*—*put*.

It should be emphasized that when talking about sense relations we are talking about meaning relations between lexical items. These relations have to be distinguished from ontological relations, notwithstanding that there is a close link between the two: “sense relations are relations between words (in a reading) based on ontological relations between the concepts that constitute the meanings of these words (in that reading)” (Nickles et al. 2007: 37). Ontologies can be conceived of as networks of cross-connected conceptualizations, with the relations holding between those conceptualizations being ontological relations. (This, admittedly, is a very broad definition, which in principle allows any conceptual relation to be subsumed under the label “ontological relation.”) Ontological relations might exist without being reflected in a language’s lexicon. An example is the concept PUT, for which there is no German lexical coding. Instead, German uses *stellen*, *setzen*, *legen*, *stecken*, which all involve an actor displaying a behavior that causes an exponent to move into a goal location (*Er stellte das Buch ins Regal* ‘He put the book on the shelf’). But they all include additional information on the orientation or position of the motion exponent in the goal location: upright (*stellen*), sitting (*setzen*), horizontal (*legen*), or partially fitting into an aperture (*stecken*). Although the respective concepts are all subordinate concepts of PUT and they thus all entertain an ontological relation to PUT, there are no sense relations in German corresponding to these ontological relations, as there is no lexical item available to express the superordinate concept PUT. Unfortunately, the distinction between lexical and ontological relations is often not clearly drawn, neither in linguistics nor in NLP and AI. The most obvious case in point is that WordNet, which was devised as a lexical resource and built on the basis of sense relations, is often treated and discussed as if it were an ontology at the same time (Fellbaum 2007: 420).

Ontologies—particularly upper-level and domain ontologies—have been developed and employed in AI and knowledge representation (e.g., Sowa 2000), and more generally in computer science, for some time. In computational linguistics, there is accordingly a branch which systematically takes advantage

* The Semantic Web is conceived of as an extension of the World Wide Web, which aims to provide conceptual descriptions and structured information on documents, in an effort to make the knowledge contained in these documents understandable by machines. Means to achieve this include semantic tagging and the use of specifically designed standards and tools such as the Resource Description

Framework (RDF), Web Ontology Language (OWL), and Extensible Markup Language (XML). For more information and pointers to references, refer to this chapter's section in the accompanying wiki.

of ontologies in the AI sense (also see Chapter 24). One of the most extensive works in this area is Nirenburg and Raskin's (2004) *Ontological Semantics*. Their ontology lists definitions of concepts for describing the meanings of lexical items of natural languages, but also for specifying the meanings of the text-meaning representations that serve as part of an interlingua for machine translation (Nickles et al. 2007). Application domains also include information extraction, question answering, human-computer dialog systems, and text summarization.

Ontological semantics is a theory of meaning in natural language and an approach to natural language processing (NLP) which uses a constructed world model, or ontology, as the central resource for extracting and representing meaning of natural language texts, reasoning about knowledge derived from texts as well as generating natural language texts based on representations of their meaning. The architecture of an archetypal implementation of ontological semantics comprises, at the most coarse-grain level of description:

- a set of static knowledge sources, namely, an **ontology**, a **fact database**, a **lexicon** connecting an ontology with a natural language and an **onomasticon**, a lexicon of names (one lexicon and one onomasticon are needed for each language);
 - **knowledge representation languages** for specifying meaning structures, ontologies, and lexicons; and
 - a set of processing modules, at the least, a semantic **analyzer** and a semantic **text generator**.
- (Nirenburg and Raskin 2004:10)

In linguistics proper, ontology-based approaches are only just becoming an explicit focus of research. In addition to ontological relations underlying sense relations and hence impacting on lexical structure, it has recently been recognized that there are many more areas of semantic research that need to take language ontologies into account. Several examples, including lexical semantics, classifiers, word classes, anaphora resolution, and the grammar of coordination, can be found in Nickles et al. (2007: 37–38).

5.3.2 Roles

While ontological relations are stable conceptual relations, there are also situation-specific relations. For example, an instance of an eventuality such as German *setzen über* ('cross a waterbody,' 'leap across an obstacle') requires a specification of its participants. Who is crossing or leaping across what? What are they deploying to do so? What is important in the relational context is that such participant specifications include the role they play in the eventuality they partake in. In semantics, these roles are referred to as semantic roles, thematic roles, or predicate-argument relations (Van Valin 2004). Semantic roles are important for the linking of the arguments to their syntactic realizations (e.g., a patient participant is realized in English as a direct object of an active voice sentence), in addition to their value in describing and representing semantics. We will not focus on the linking of arguments in the following, the interested reader may want to consult, for instance, Dowty (1991), Goldberg (1995), Van Valin and LaPolla (1997), Van Valin (1999), and Bowerman and Brown (2008).

The number and definition of semantic roles are still subject to controversial discussions. Initiated by Gruber (1965) and Fillmore (1968), there have been two opposing trends in linguistics with regard to semantic roles. Localistic approaches, descending from Gruber, work with a small number of concrete, spatial relationships, such as *goal* and *source*, which they see as the foundation for other, more abstract roles. For example, an experiencer, intentional agent, or originating location would all be assigned the source role. Non-localistic approaches, descending from Fillmore, work with a much larger number of

roles, which can be tailored to specific verbal subclasses. The FrameNet project (Ruppenhofer et al. 2006),

TABLE 5.2 Semantic Roles and Their Conventional Definitions

Role	Description
<i>agent</i>	a wilful, purposeful instigator of an action or event
<i>effector</i>	the doer of an action, which may or may not be wilful or purposeful
<i>experiencer</i>	a sentient being that experiences internal states, such as perceivers, cognizers, and emoters
<i>instrument</i>	a normally inanimate entity manipulated by an agent in carrying out an action
<i>force</i>	somewhat like an instrument, but it cannot be manipulated
<i>patient</i>	a thing that is in a state or condition, or undergoes a change of state or condition
<i>theme</i>	a thing which is located or is undergoing a change of location (motion)
<i>benefactive</i>	the participant for whose benefit some action is performed
<i>recipient</i>	someone who gets something (recipients are always animate or some kind of quasi-animate entity)
<i>goal</i>	destination, which is similar to recipient, except that it is often inanimate
<i>source</i>	the point of origin of a state of affairs
<i>location</i>	a place or a spatial locus of a state of affairs
<i>path</i>	a route

Source: Van Valin, R.D. and LaPolla, R.J., *Syntax: Structure, Meaning and Function*, Cambridge University Press, Cambridge, U.K., 1997, 85–89.

for example, which is based on Fillmore’s frame semantics, uses a very large number of semantic roles (so-called frame elements), which are defined separately for each frame:

Although something as straightforward as ‘Agent’ will in one frame have very much in common with the ‘Agent’ in another unrelated frame — based on what we all agree to be true about agents — it is also true that the Agent role in each frame is operating within a unique context.

For example, the Agent in the Accomplishment frame, besides exerting his conscious will, is specifically involved in achieving a particular Goal that requires some amount of time and effort; in the Adjusting frame, this Agent is controlling Parts to bring about some effect. These facts about these Agents and their contexts are defined by the frame, but it is not possible to completely divorce these facts from the semantics of the ‘do-er,’ called an Agent for convention and convenience’s sake. In other words, we could just as easily call him the Accomplisher or the Adjuster.*

Austere and abstract localism, on the one hand, and the FrameNet approach, on the other, represent extremes on a spectrum. In practice, most linguists nowadays tend to employ a set of a dozen or so semantic roles, regarding them as somewhat fuzzy, prototypical notions rather than fixed absolute categories. Van Valin and LaPolla (1997) give the non-exhaustive list shown in Table 5.2. Nevertheless, there is no consensus on which prototypical role notions are to be included and how they should be defined. Much is left to intuition and interpretation.

Furthermore, although a set of prototypical role notions seems desirable, such a list would not suffice to model all eventities. To return to the example of *setzen über* (Schalley 2004a: 315–322), it clearly involves an agent (the wilful purposive instigator), an instrument (the entity manipulated by the agent to carry out the action), and a location or ground (the place or locus where the leaping or crossing occurs). Yet, as mentioned above, the ground also needs to be an obstacle that has to be overcome, and

therefore an event-related characteristic needs to be added to the ground specification. In other words, if semantic roles are specified using prototypical role notions, some mechanism needs to be provided to allow for additional specifications on the role of a participant. The UER system (Section 2.5) provides such mechanisms by defining prototypical participant roles in participant class specifications, and allowing additional characteristics to be expressed via attributes. In FrameNet, the Core Frame Elements can

* This quotation is taken from FrameNet FAQs, 'Are Frame Elements unique across frames?'; <http://framenet.icsi.berkeley.edu/index.php?option=com_content&task=view&id=19038&Itemid=49>. We have taken the liberty of correcting a number of typographical errors.

be further defined using ‘semantic types’ (STs). “For example, in the Chatting frame, the Interlocutor definitions take the ST ‘Human,’ thereby excluding non-human agents as Core participants in the frame.”*

Not all systems accept the existence of semantic roles as independent entities, either in an absolute or prototypical sense. In the NSM system, the relevant generalizations are seen to emerge from top-level components of the meaning structures of individual verbs. At this level, many verbs are based on the semantic primes *DO*, *HAPPEN*, and *WANT*, the first two of which have several alternative valency frames. Rather than saying that complex physical activity verbs like *cut*, *chop*, *dig*, and *grind* involve an agent, a patient, and an instrument, for example, in the NSM system these verbs are analyzed as sharing the top-level components (lexico-syntactic frame) shown in (6a) below. Likewise, verbs of bodily locomotion like *walk*, *run*, and *swim* are assigned the frame in (6b) (Goddard and Wierzbicka 2009). Additional specifications on the nature of the participants or their roles can be spelt out later in the explication.

(6a) NSM lexico-syntactic frame for complex physical activities

- i. someone X was doing something to something Y with something else Z for some time
- ii. because of this, something was happening at the same time to this something Y, as this someone wanted

(6b) NSM lexico-syntactic frame for bodily locomotion

- i. someone X was doing something somewhere with parts of his/her body for some time
- ii. because of this, this someone’s body was moving at the same time in this place, as this someone wanted

Another approach that considers semantic roles not to be independent entities is Jackendoff’s (1990, 2002) Conceptual Semantics. Jackendoff (1990: 127) explicitly identifies semantic roles as the argument slots of abstract basic predicates.

5.4 Fine-Grained Lexical-Semantic Analysis: Three Case Studies

In this section, we present three case studies of fine-grained lexical-semantic analysis. The framework is the NSM method (Section 5.2.4), but to a considerable extent the findings can be repackaged into different models.

5.4.1 Emotional Meanings: “Sadness” and “Worry” in English and Chinese

Formal semantics has had little to say about the meanings of emotion words, but words like these are of great importance to social cognition and communication in ordinary language use. They may be of special importance to NLP applications connected with social networking and with machine–human interaction, but they differ markedly in their semantics from language to language. In this section, we will illustrate with contrastive examples from English and Chinese.

Consider first the difference between English *sad* and *unhappy*. Conventional dictionaries may make them seem virtually synonymous, but this does not tally with the intuitions of native speakers or with the evidence of usage. There are contexts in which one could use one word but not the other, as shown in (7), or where the choice of one word or the other would lead to different implications, as suggested by the examples in (8) (Wierzbicka 1999: 60–63, 84–85).

* FrameNet FAQs, ‘Are Frame Elements unique across frames?’; http://framenet.icsi.berkeley.edu/index.php?option=com_content&task=view&id=19038&Itemid=49. It should be noted that the attribute ‘human’ is not, strictly speaking, a role characteristic, but rather a selectional restriction on the participants. Although important in semantic

representation, selectional restrictions should, for the sake of precise semantic modeling, be kept separate from role characterizations.

- (7) *I miss you a lot at work..... I feel so sad (*unhappy) about what's happening to you* [said to a colleague in hospital who is dying of cancer].
 (8a) *I was feeling unhappy at work* [suggests dissatisfaction].
 (8b) *I was feeling sad at work* [suggests depression, sorrow, etc.].

NSM explications for emotion meanings work by linking a feeling (usually a good feeling or a bad feeling) with a prototypical cognitive scenario which serves as a kind of reference situation. Explication [B] below is for the expression *feel sad*. Notice that the content of the scenario, in section (b) of the explication, is given in the first-person. It involves awareness that 'something bad happened' (not necessarily to me), which I didn't want to happen but am prepared to accept, in the sense of recognizing that I can't do anything about it (an attitude akin to resignation). Explication [B] is compatible with the wide range of use of *sad*; for example, that I may *feel sad* on hearing that my friend's dog has died or when thinking about some bickering in my workplace. *Unhappy* has a more personal character than *sad*: one feels *unhappy* because of bad things that have happened to one personally, things that one did not want to happen. The attitude is not exactly active, but it is not passive either. It suggests something like dissatisfaction, rather than resignation. These properties are modeled in explication [C]. Notice that there is an extra final component (c), saying that the experiencer actually had thoughts like those identified in the cognitive scenario. In other words, feeling *unhappy* implies a more cognitively active state than feeling *sad*. (While one can say *I feel sad, I don't know why*, it would be a little odd to say *I feel unhappy, I don't know why*.)

[B] Semantic explication for *Someone X felt sad*

- a. someone X felt something bad
 like someone can feel when they think like this:
- b. "I know that something bad happened
 I don't want things like this to happen
 I can't think like this: I will do something because of it now
 I know that I can't do anything"

[C] Semantic explication for *Someone X felt unhappy*

- a. someone X felt something bad
 like someone can feel when they think like this:
- b. "some bad things happened to me
 I wanted things like this not to happen to me
 I can't not think about it"
- c. this someone felt something like this, because this someone thought like this

For speakers of English, states of mind like 'feeling sad' and 'feeling unhappy' may seem so natural that one could assume they would have matching words in all languages, but this is very far from being the case. Ye (2001) shows that Chinese lacks a precise equivalent to English "sadness" (cf. Russell and Yik 1996). One possible match would be *ai*, but this is strongly linked with mourning (hence, closer to English *sorrow*). The two other candidates are *bei* and *chou* (tone markings omitted), but each differs significantly from English *sad*. *Bei* is more intense and has a tragic, fatalist tone, suggesting powerlessness before the laws of nature, the inevitability of ageing and death, etc. *Chou* lacks this existential gravitas and is a more common everyday word than *bei*, but *chou* is at least as close to English *worried*, as it is to *sad*. *Chou* is focused on the experiencer's present situation. One experiences this feeling when confronting a personal predicament that is forced on one by the circumstances, "leaving the experiencer caught in a dilemma — wanting to overcome a difficult situation, yet not finding a solution" (Ye 2001: 379). The

experiencer, moreover, cannot stop thinking about it, creating a link with English *preoccupied*. Ye (2001) proposes the following explications (slightly modified):

[D] Semantic explication for *Someone X felt bei* [Chinese]

- a. someone X felt something very bad
like someone can feel when they think like this:
- b. “something bad happened now
I know that after this good things will not happen anymore
I don’t want things like this to happen
I want to do something if I can
I know that I can’t do anything
because I know that no one can do anything when things like this happen”
- c. this someone felt something like this, because this someone thought like this

[E] Semantic explication for *Someone X felt chou* [Chinese]

- a. someone X felt something bad
like someone can feel when they think like this:
- b. “something bad is happening to me
before this, I did not think that this would happen to me
I don’t want things like this to happen to me
because of this, I want to do something if I can
I don’t know what I can do
I can’t not think about this all the time”
- c. this someone felt something like this, because this someone thought like this

It seems clear that Chinese *bei* and *chou* have no close equivalents in English, and conversely, that *sad* and *unhappy* have no close equivalents in Chinese. Similar demonstrations of semantic nonequivalence could be adduced for many other languages and for many other areas of the mental states lexicon (Russell 1991; Goddard 1996a; Wierzbicka 1999, 2004; Harkins and Wierzbicka 2001; Enfield and Wierzbicka 2002; Schalley and Khelentzos 2007).

5.4.2 Ethnogeographical Categories: “Rivers” and “Creeks”

Words for landforms and landscape features — like English *desert*, *mountain*, and *river* — might seem an unlikely domain for complex lexical semantics or for extensive crosslinguistic variation, since it seems at first blush that their referents are objective physical entities. But things are not that simple. Unlike as in the biological world, where to a large extent a structure of kinds (species) is “given” by nature, the demarcation between different categories of geophysical referents is often far from clear-cut (compare *mountain* and *hill*, *river* and *stream*, *lake* and *pond*). Languages differ considerably in the nature of the distinctions they recognize (Burenhult and Levinson 2008). Sometimes these differences are related to the nature of the landforms themselves, but categorization of the landscape also has a human dimension. It reflects anthropocentric concerns, and these concerns can vary with the culture as well as with the physical world.

The problem has been noted by researchers in geospatial data information systems, as well as by geographers and linguists (Mark and Turk 2003, 2004). Some have called for the creation of a new field of study into naïve geography: “a set of theories that provide the basis for designing future geographic information systems (GIS) that follow human intuition and are, therefore, easily accessible to a large range of users” (Egenhofer and Mark 1995). From a computational point of view, questions like the following arise (Scharl 2007; Oosterom and Zlatanova 2008; Stock 2008): What is the optimal ontology for geospatial databases? What set of place categories or semantic descriptions should be used to annotate and index them? How can querying systems be implemented that will be responsive to the expectations of user communities with different native languages? How can interoperability be achieved between

heterogeneous Web services and data sets in the geographic domain? Is it possible to devise systems for

extracting and compiling geographical information from Web-based documents, for example, online news reports?

We now report on lexical-semantic analysis of geographical categories being undertaken by Bromhead (2008), taking our examples from “elongated” hydrological features. Explication [F] for English *river* claims that a *river* is conceptualized as ‘a place of one kind’: a place with a lot of water, a long, big place that can be a potential barrier at times (components (b)–(e)). The water, furthermore, is constantly moving and can be thought of as, so to speak, traveling from a distant source to a destination far away (components (f) and (g)).

[F] Semantic explication for *a river*

- a. a place of one kind
- b. there is a lot of water [M] in places of this kind
- c. places of this kind are long [M] places
- d. places of this kind are big places
- e. because it is like this, at many times when someone is on one side of a place of this kind, if this someone wants to be on the other side, this someone can’t be on the other side after a short time
- f. the water [M] in places of this kind is moving at all times
- g. when someone is somewhere on one side of a place of this kind, this someone can think like this:
 “some time before this, this water [M] was in a place far from here
 some time after this, this water [M] will be in another place far from here”

The comparable explication for *stream* is distinguished from that of *river* in terms of the smaller size of the place and lesser quantity of water (not only moving, but moving quickly), and also by lacking components corresponding to (e) and (g) above. These intuitively plausible differences correlate with collocational preferences of the two words; for example, the acceptability contrast between expressions like *great river* and ?*great stream*, and the much higher frequency of *little stream*, as compared with *little river*.

In Australian English, the word *creek* is extremely common, and like *stream* it refers to a smaller water feature than a *river*. What then is the difference between *creek* and *stream*? The major differences are connected with the fact that in the world’s driest continent the water flow in most watercourses is not reliable. Typically the water level in a *creek* fluctuates a lot and, in times of drought may dry up entirely. Consistent with this, collocations of the word *creek* with expressions like *dries up* and *dried up* are quite common. Likewise, the expression *creekbed* is common, referring to the dry surface (not to the bottom of a full creek, as does the comparable form *riverbed*). Consider explication [G]. For *creek*, both the presence of water in the said location and its movement is qualified by ‘at many times.’ The variability of the water level is implied by the additional component (c).

[G] Semantic explication for *a creek* [Australian English]

- a. a place of one kind
- b. at many times there is water [M] in parts of places of this kind
- c. at some times there is water [M] in all parts of places of this kind
- d. places of this kind are long [M] places
- e. places of this kind are not big places
- f. at many times the water [M] in places of this kind is moving

Consider now the word *karu* from the Central Australian Aboriginal language Yankunytjatjara. In Central Australia, a *karu* is usually dry, and as such would be normally referred to in English as a *creekbed*, but *karu* makes no distinction between what English describes differently as either a *creek* or a *creekbed*. As Mark et al. (2007: 8) observe: “in arid and semi-arid regions, the distinction between topographic and hydrological features quite literally dries up.” Some naturally occurring examples of *karu* (cf. Goddard

1996b) are as follows:

(9a) *Ka karungka uru pulka ukalingangi*. ‘There was a lot of water flowing in the *karu*.’ (9b) *Apara tjuta ngarala wanani karungka*. ‘River gums line the *karu*.’

(9c) *Haast’s Bluff-ala ngayulu itingaringu, karungka, manta tjulangka*. ‘I was born at Haast’s Bluff, in the *karu*, where the ground is soft.’

Explication [H] contains several components that are the same or similar to those in English *creek*, but there are significant differences. Water is said to be present only ‘at some times,’ and, more dramatically, there is a description of what the ground is like in a *karu*, that is, lower than the surroundings and distinctively soft. A final component adds the detail, very salient in the traditional desert lifestyle, that water is often to be found below the ground in a *karu* (it can be obtained by digging in a so-called soakage).

[H] Semantic explication for *karu* (‘creekbed, creek’) [Yankunytjatjara]

- a. a place of one kind
- b. at some times there is water [M] in places of this kind
- c. places of this kind are long [M] places
- d. the ground [M] in a place of this kind is below the ground [M] on both sides of a place of this kind
- e. it is not like the ground [M] in other places, it is soft [M]
- f. when there is water [M] in places of this kind, at some times this water [M] is moving
- g. at many times there is water [M] in some places below the ground [M] in places of this kind

Semantic analysis along similar lines can be used to spell out the different conceptualizations behind terms for all sorts of landscape (and seascape) features. As well as providing plausible semantic structures that can be linked with collocational and other linguistic evidence, such explications offer a potential solution to another problem that has sometimes been identified for landscape features, namely, the existence of a certain indeterminacy and overlap in the referential ranges of related words. Going back to *river*, *stream*, and *creek*, for example, the explications account for why certain physical referents can be open to being described by any of these three terms. The difference is not necessarily a matter of physical dimensions, but can be determined by how the user is thinking about the referent at the time.

5.4.3 Functional Macro-Categories

Functional macro-category words, also called collective nouns (e.g., Mihatsch 2007), pose interesting semantic challenges in themselves, not least because they tend to vary across languages to a much greater extent than do true taxonomic words. In addition, their semantics have a direct relevance to the architecture of semantic networks: semantic networks are typically organized in a hierarchical fashion, with classificatory relationships indicated by the “is-a” relation, which is often assumed to correspond essentially to ‘is a kind of.’ Higher level nodes are typically labeled indifferently with either taxonomic or with functional macro-category words. To put it another way, the “is-a” relationship sometimes corresponds to a genuine taxonomic one and sometimes does not, leading to inconsistent results and mismatches with the intuitions of ordinary speakers (Brachman 1983; cf. Wisniewski et al. 1996; Cruse 2002; Veres and Sampson 2005). In this section, we will concentrate on English and discuss the semantic differences between three types of classificatory relationships that can be exemplified as shown in Table 5.3.

TABLE 5.3 Three Different Types of Classificatory Relationships in English

True Taxonomic	“Plural-Mostly” Functional	Singular-Only Functional-Collective
Category Words	Macro-Category Words	Macro-Category Words

<i>birds—sparrow, wren, eagle, ...</i>	<i>vegetables—carrots, peas, celery, ...</i>	<i>furniture—table, chair, bed, ...</i>
<i>fish—trout, tuna, bream, ...</i>	<i>herbs—basil, oregano, rosemary, ...</i>	<i>cutlery—knife, fork, spoon, ...</i>
<i>animal—dog, cat, horse, ...</i>	<i>cosmetics—lipstick, powder, mascara, ...</i>	<i>jewelry—ring, earring, necklace, ...</i>

The relationships illustrated in the left-hand column of Table 5.3 are genuinely taxonomic. For example, *birds* are ‘living things of one kind’ and this kind is understood to have various sub-kinds. The meaning of the words *sparrow*, *wren*, and *eagle*, for example, all contain the semantic molecule ‘bird [M].’ The relationships exemplified in the other two columns are quite different, both semantically and in their grammatical behavior. Though words like *vegetables* and *furniture* may express classification in a broad sense, it is not in terms of a taxonomic hierarchy. Words like *vegetables*, *herbs*, *cosmetics*, *furniture*, *cutlery*, and *jewelry* do not designate ‘things of one kind,’ but ‘things of many kinds’ unified by factors such as shared functions, contiguity, and similar origins (Wierzbicka 1985, esp. Chap. 4, pp. 258–354; Goddard 2009).*

“Plural-mostly” functional macro-category words (e.g., *vegetables*, *cosmetics*, *herbs*, *cereals*, *drugs*) are so called because they occur predominantly in the plural, other than when they are bare stems in compounds, for example, *vegetable soup*, *cosmetic surgery*. Singular occurrences are mostly in predicate position with the indefinite article *a*, in a generic construction, for example, *Spinach is a green vegetable*; while singulars with the definite article are rare and anomalous (?*the vegetable*, ?*the cosmetic*). Their most distinctive semantic property is that when counting the referents of “plural-mostly” functional categories, such as *vegetables*, *cosmetics*, etc., one counts not individual items but kinds. For example, *There were only two vegetables on the plate* means ‘only vegetables of two kinds.’

A semantic explication for *vegetables* is given in [I]. As mentioned, it begins with the component ‘things of many kinds.’ Component (b) is an exemplar component, mentioning several of the most salient kinds (cf. Battig and Montague 1969; Rosch 1973; Rosch et al. 1976), which, needless to say, are highly culture-specific. Component (c) sets out the salient properties shared by the various kinds, presented not as inherent properties but in terms of how people think about them — as edible items requiring some preparation (cooking, peeling, washing, etc.), that come from things that people cultivate and that grow close to the ground. They are not (thought of as) sweet. Component (d) says that these shared construals allow people to see things of these different kinds as ‘like things of one kind.’ This makes the macro-category “quasi-taxonomic.” The semantic structure of other “plural-mostly” macro-category words, such as *herbs*, *cereals*, *cosmetics*, and *drugs* follow the same basic template.

[I] Semantic explication for *vegetables*

- a. things of many kinds
- b. carrots [M] are things of one of these kinds, peas [M] are things of one of these kinds
- c. people can think about things of all these kinds like this:
 - people can eat [M] things of these kinds, if someone does some things to them beforehand
 - before people can eat [M] things of this kind, they are parts of some other things
 - people do things in some places because they want these things to grow [M] in these places
 - when things of these kinds grow [M] in a place, they are near the ground [M]
 - things of this kind are not sweet [M]
- d. because people can think about things of all these kinds like this, people can think that they are like things of one kind

Singular-only functional-collective category words (e.g., *furniture*, *cutlery*, *clothing*, *crockery*, *jewelry*) are invariably singular in form (**furnitures*, **clothings*, etc.), but, unlike normal singular nouns, they cannot take an indefinite singular article or the word *one* (**a/one furniture*, **a/one clothing*). A distinctive property is that they are generally compatible with the *item(s) of X* or *piece(s) of X* construction. Although grammatically singular, these nouns can be objects of verbs that imply a multiplex object, for example, *laying out the cutlery*, *re-arranging the furniture*, *sorting through the clothing*. Many are compatible with

* The source of the confusion is no doubt the assumption that set inclusion (a referential, extensionalist relationship) equates to a 'kind of' relationship (an intensional, conceptual relationship), but this assumption is flawed. Every policeman, for example, is somebody's son, but it doesn't follow that a policeman is conceptualized as a 'kind of son' (Wierzbicka 1985: 259).

phrases like *a collection of X*, *a range of X* or *a set of X*. Words like *furniture*, *cutlery*, *clothing*, *crockery*, and the like, tend to imply “unity of place,” that is, they designate things of different kinds that are expected to be used in heterogeneous groups of items all put in the same place.

A sample explication is given in [J]. As with other functional macro-category explications, it begins with ‘things of many kinds,’ followed by a component stating the most salient exemplar kinds (in the case of *furniture*: tables and chairs). Next comes a set of shared properties, including location in people’s homes (and other similar places), serving the purpose, roughly speaking, of bodily comfort and convenience. There is also a “moveability” component. On account of these shared properties, it is claimed, when things of these different kinds are in the one place, people can think that they are ‘like one thing.’ This semantic property is linked with singular number status of words of this type. Other singular-only functional macro-category words, such as *cutlery*, *crockery*, *clothing*, and *jewelry*, follow the same basic semantic template.

[J] Semantic explication for *furniture*

- a. things of many kinds
- b. tables [M] are things of one of these kinds, chairs [M] are things of one these kinds
- c. people can think about things of all these kinds like this:
 - there are many things of these kinds in people’s homes [M]
 - there are many things of these kinds in other places where people want to be for some time
 - these things are like parts of these places
 - people want things of these kinds to be in these places because they don’t want to feel something bad in their bodies when they do some things in these places for some time
 - if something of one of these kinds is somewhere in a place at some time, at another time it can be somewhere else in this place if this someone wants it
- d. because people can think about things of all these kinds like this, when things of these kinds are in one place, people can think that they are like one thing

If the semantic analyses in [I] and [J] are correct in their main outlines, in order to be faithful to human cognitive organization, it is problematical to use functional macro-categories as higher level nodes in semantic networks, as if they were taxonomic categories. They could be more appropriately used as attribute features, though it has to be borne in mind that such categories are highly variable across languages.

5.5 Prospectus and “Hard Problems”

Despite declining “syntacticocentrism” (Jackendoff 2002: 107–111), semantics—especially lexical semantics—remains the poor cousin of linguistics. There are relatively few specialists, especially with additional crosslinguistic or NLP expertise.

The complexity of lexical semantics and the extent of lexical-semantic differences across languages have been vastly underestimated by most NLP practitioners and, for that matter, by most linguists. To some extent, this underestimation is due to the tendency of twentieth century work to concentrate on truth-conditional and referentialist approaches to meaning. Though there are in fact considerable divergences between languages in words that refer to concrete physical phenomena (including biological species), the extent of differences is much greater in relation to more cultural, more subjective words—words whose meanings are, so to speak, creatures of the mind. Such meanings are crucial to ordinary social cognition and to ordinary language use and interpretation. To mention two more categories, there are culture-specific social ideals, such as Japanese *wa* and Chinese *xiào*, usually clumsily rendered into English as “unity, harmony” and as “filial piety,” respectively (Ho 1996; Wierzbicka 1997: 248–254; Goddard 2005:

84–88); and there are culture-specific social categories, such as English *friend*, Russian *drug* ‘trusted friend,’ Chinese *zìjǐ rén* ‘insider, one of us’ and *wàirén* ‘outsider’ (Wierzbicka 1997: 32–84; Ye 2004).

Another significant problem are abstract nouns such as English *rights*, *security*, *dialogue*, *trauma*, and *experience*, which lack exact equivalents in many of the world's languages (Wierzbicka 2006, 2009b).

Despite the tremendous advances in computational power and improvements in corpus linguistics, the prospectus is not particularly good for next-generation NLP applications that would depend on high-detail semantic analysis; for example, in information extraction from unstructured text (see Chapter 21), machine translation, and the Semantic Web. Many common semantic phenomena are likely to remain computationally intractable for the foreseeable future. Rough-and-ready semantic processing (partial text understanding), especially in restricted domains and/or in restricted "sublanguages" offer more promising prospects but we have to be realistic about what is achievable, even with next-generation technologies.

A particular problem is that standard resources such as WordNet and GlobalWordNet cannot support fine-grained semantic analysis even for English, let alone for other major languages of the world. Major controlled vocabulary dictionaries such as Collins Cobuild have their uses, but they too are not adequate to the task (Hoelter 1999; Guenther 2008). Though corpus techniques and other computational tools are providing valuable new resources and heuristics, there are no quick and easy automatic methods for generating detailed lexical-semantic analyses. They must be hand built by experienced human analysts and it will be a long time before even a decent fragment of "ordinary English" (to speak with Montague 1973) has been adequately analyzed.

It might be a practical proposition to specify moderately sized controlled auxiliary languages, such as, in the case of English, Formalized English and Frame-CG (cf. Martin 2002) or an improved "Globish" (Nerrière 2004), and to hand build complete semantically analyzed lexicogrammars for them. But such auxiliary languages would be radically constrained compared with their ordinary unconstrained counterparts.

Above all, if progress is to be made, researchers in NLP and linguistic semantics must give serious consideration to the question of standards, which play a prominent role in computer science. It would not be possible, or even desirable, to standardize working methods among all the multifarious research groups and communities involved, but there needs to be some collective attention to the problem of ensuring the longevity and comparability of research results and interoperability of systems. The enduring presence of the predicate calculus (despite its inadequacies) is largely a testimony to the value of widely known and understood notations. We venture to suggest that intuitive systems such as those based on reductive paraphrase in natural language (for verbal representation) and/or on industry-wide standards such as UML (for object-oriented representation) offer the best potential to meet this need.

Acknowledgments

We are grateful to Bonnie Webber, Nitin Indurkha, and Csaba Verés for comments on an earlier version of this chapter. This research was supported by the Australian Research Council.

References

- Andrews, A. D. 2006. Semantic composition for NSM, using LFG + glue. In *Selected Papers from the 2005 Conference of the Australian Linguistics Society*, Melbourne, Australia, K. Allan (ed.). <http://www.als.asn.au>
- Apresjan, J. D. 1992. *Lexical Semantics: User's Guide to Contemporary Russian Vocabulary*. Ann Arbor, MI: Karoma. [Orig. published 1974 as *Leksiceskaja Semantika—Sinonimiceskie Sredstva Jazyka*. Moskva, Russia: Nauka.]

Apresjan, J. D. 2000. *Systematic Lexicography* (Trans. K. Windle). Oxford, U.K.: Oxford University Press. Asher, N. 1993. *Reference to Abstract Objects in Discourse*. Dordrecht, the Netherlands: Kluwer.

- Asher, N. and A. Lascarides 2003. *Logics of Conversation*. Cambridge, U.K.: Cambridge University Press.
- Barwise, J. and J. Perry. 1983. *Situations and Attitudes*. Cambridge, MA: MIT Press.
- Battig, W. F. and W. E. Montague. 1969. Category norms for verbal items in 56 categories: A replication and extension of the Connecticut category norms. *Journal of Experimental Psychology Monograph* 80(3):1–46.
- Behrens, L. 1998. Polysemy as a problem of representation—‘representation’ as a problem of polysemy. Review article on: James Pustejovsky. *The Generative Lexicon*. *Lexicology* 4(1):105–154.
- Blackburn, P. and J. Bos. 1999. *Working with Discourse Representation Theory: An Advanced Course in Computational Semantics*. Stanford, CA: CSLI Press. Manuscript. <http://homepages.inf.ed.ac.uk/jbos/comsem/book2.html>
- Blackburn, P. and J. Bos. 2005. *Representation and Inference for Natural Language: A First Course in Computational Semantics*. Stanford, CA: CSLI Press.
- Bowerman, M. and P. Brown (eds.). 2008. *Crosslinguistic Perspectives on Argument Structure*. New York, NY/London, U.K.: Taylor & Francis.
- Brachman, R. 1983. What IS-A is and isn’t: An analysis of taxonomic links in semantic networks. *IEEE Computer* 16(10):30–36.
- Bromhead, H. 2008. Ethnogeographical classification in Australia. *Paper presented at the 2008 Annual Meeting of the Australian Linguistics Society*. Sydney, Australia, July 3, 2008.
- Burenhult, N. and S. C. Levinson (eds.). 2008. Language and landscape: A cross-linguistic perspective (Special Issue). *Language Sciences* 30(2/3):135–150.
- Cann, R. 1994. *Formal Semantics. An Introduction*. Cambridge, U.K.: Cambridge University Press.
- Croft, W. and D. A. Cruse. 2003. *Cognitive Linguistics*. Cambridge, U.K.: Cambridge University Press.
- Cruse, D. A. 1986. *Lexical Semantics*. Cambridge, U.K.: Cambridge University Press.
- Cruse, D. A. 2002. Hyponymy and its varieties. In *The Semantics of Relationships: An Interdisciplinary Perspective*, R. Green, C. A. Bean, and S. H. Myaeng (eds.), pp. 3–21. Dordrecht, the Netherlands: Kluwer.
- Cruse, D. A. 2004. *Meaning in Language: An Introduction to Semantics and Pragmatics*, 2nd edn. Oxford, U.K.: Oxford University Press.
- Dalrymple, M. 2001. *Lexical Functional Grammar*. San Diego, CA: Academic Press.
- Dowty, D. R. 1991. Thematic proto-roles and argument selection. *Language* 67(3):547–619.
- Dowty, D. R., R. E. Wall, and S. Peters. 1981. *Introduction to Montague Semantics*. Dordrecht, the Netherlands: Reidel.
- Egenhofer, M. J. and D. M. Mark. 1995. Naïve geography. In *Spatial Information Theory: A Theoretical Basis for GIS*. In *Proceedings of the International Conference COSIT’95*, Semmering, Austria, September 21–23, 1995, A. Frank and W. Kuhn (eds.). *Lecture Notes in Computer Science* 988:1–15. Berlin, Germany: Springer-Verlag.
- van Eijck, J. 2006. Discourse representation theory. In *Encyclopedia of Language and Linguistics*, 2nd edn, Vol. 3, K. Brown (ed.), pp. 660–669. Oxford, U.K.: Elsevier.
- Enfield, N. J., and A. Wierzbicka (eds.). 2002. The body in the description of emotion (Special Issue). *Pragmatics and Cognition* 10(1).
- Fauconnier, G. 1985. *Mental Spaces*. Cambridge, MA: MIT Press.

- Fellbaum, C. 1998a. A semantic network of English: The mother of all WordNets. In *EuroWordNet: A Multilingual Database with Lexical Semantic Networks*, P. Vossen (ed.), pp. 209–220. Dordrecht, the Netherlands: Kluwer.
- Fellbaum, C. (ed.). 1998b. *WordNet: An Electronic Lexical Database*. Cambridge, MA: MIT Press.
- Fellbaum, C. 2007. The ontological loneliness of verb phrase idioms. In *Ontolinguistics: How Ontological Status Shapes the Linguistic Coding of Concepts*, A. C. Schalley and D. Zaefferer (eds.), pp. 419–434. Berlin, Germany: Mouton deGruyter.

- Fillmore, C. J. 1968. The case for case. In *Universals in Linguistic Theory*, E. Bach and R. T. Harms (eds.), pp. 1–88. New York, NY: Holt, Rinehart & Winston.
- Fried, M. and J.-O. Östman. 2004. Construction grammar. A thumbnail sketch. In *Construction Grammar in Cross-Language Perspective*, M. Fried and J.-O. Östman (eds.), pp. 11–86. Amsterdam, the Netherlands: Benjamins.
- Gamut, L. T. F. 1991. *Logic, Language and Meaning*, 2 Vols. Chicago, IL/London, U.K.: University of Chicago Press.
- Geeraerts, D. 2002. Conceptual approaches III: Prototype theory. In *Lexikologie/Lexicology. Ein internationales Handbuch zur Natur und Struktur von Wörtern und Wortschätzen. [An International Handbook on the Nature and Structure of Words and Vocabularies]*, Vol. I., D. A. Cruse, F. Hundsnurscher, M. Job, and P. R. Lutzeier (eds.), pp. 284–291. Berlin, Germany: Mouton de Gruyter.
- Geurts, B. and D. I. Beaver. 2008. Discourse representation theory. In *The Stanford Encyclopedia of Philosophy* (Winter 2008 Edition), E. N. Zalta (ed.), Stanford University, Stanford, CA. <http://plato.stanford.edu/archives/win2008/entries/discourse-representation-theory/>
- Goddard, C. 1996a. The “social emotions” of Malay (Bahasa Melayu). *Ethos* 24(3):426–464.
- Goddard, C. 1996b. *Pitjantjatjara/Yankunytjatjara to English Dictionary*, revised 2nd edn. Alice Springs, Australia: Institute for Aboriginal Development.
- Goddard, C. 1998. *Semantic Analysis: A Practical Introduction*. Oxford, U.K.: Oxford University Press.
- Goddard, C. 2000. Polysemy: A problem of definition. In *Polysemy and Ambiguity. Theoretical and Applied Approaches*, Y. Ravin and C. Leacock (eds.), pp. 129–151. Oxford, U.K.: Oxford University Press.
- Goddard, C. 2002. Overcoming terminological ethnocentrism. *IIAS Newsletter* 27:28 (International Institute for Asian Studies, Leiden, The Netherlands). http://www.iias.nl/nl/27/IIAS_NL27_28.pdf
- Goddard, C. 2005. *The Languages of East and Southeast Asia: An Introduction*. Oxford, U.K.: Oxford University Press.
- Goddard, C. 2006. Ethnopragmatics: A new paradigm. In *Ethnopragmatics: Understanding Discourse in Cultural Context*, C. Goddard (ed.), pp. 1–30. Berlin, Germany: Mouton de Gruyter.
- Goddard, C. 2007. Semantic molecules. In *Selected Papers of the 2006 Annual Meeting of the Australian Linguistic Society*, I. Mushin and M. Laughren (eds.), Brisbane, Australia. <http://www.als.asn.au>
- Goddard, C. 2009. Vegetables, furniture, weapons: Functional macro-categories in the English lexicon. *Paper Presented at the 2009 Annual Meeting of the Australian Linguistics Society*, Melbourne, Australia, July 9–11, 2009.
- Goddard, C. (ed.) 2008. *Cross-Linguistic Semantics*. Amsterdam, the Netherlands: John Benjamins.
- Goddard, C. and A. Wierzbicka. 2009. Contrastive semantics of physical activity verbs: ‘Cutting’ and ‘chopping’ in English, Polish, and Japanese. *Language Sciences* 31:60–96.
- Goddard, C. and A. Wierzbicka (eds.). 2002. *Meaning and Universal Grammar—Theory and Empirical Findings*, 2 Vols. Amsterdam, the Netherlands: John Benjamins.
- Goldberg, A. 1995. *Constructions: A Construction Grammar Approach to Argument Structure*. Chicago, IL: University of Chicago Press.
- Gruber, J. S. 1965. Studies in lexical relations. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA.

Guenther, F. 2008. *Electronic Dictionaries, Tagsets and Tagging*. Munich, Germany: Lincom.

Gutiérrez-Rexach, J. (ed.). 2003. *Semantics. Critical Concepts in Linguistics*, 6 Vols. London, U.K.: Routledge.

Harkins, J. and A. Wierzbicka (eds.). 2001. *Emotions in Crosslinguistic Perspective*. Berlin, Germany: Mouton de Gruyter.

- Heim, I. 1982. The semantics of definite and indefinite noun phrases. PhD thesis, University of Massachusetts, Amherst, MA.
- Heim, I. 1983. File change semantics and the familiarity theory of definiteness. In *Meaning, Use and Interpretation of Language*, R. Bäuerle, C. Schwarze, and A. von Stechow (eds.), pp. 164–189. Berlin, Germany: Walter de Gruyter. [Reprinted in Gutiérrez-Rexach (2003) Vol. III, pp. 108–135].
- Ho, D. Y. F. 1996. Filial piety and its psychological consequences. In *The Handbook of Chinese Psychology*, M. H. Bond (ed.), pp. 155–165. Hong Kong, China: Oxford University Press.
- Hoelter, M. 1999. *Lexical-Semantic Information in Head-Driven Phrase Structure Grammar and Natural Language Processing*. Munich, Germany: Lincom.
- Huang, C.-R., N. Calzolari, A. Gangemi, A. Lenci, A. Oltramari, and L. Prévot (eds.). In press. *Ontology and the Lexicon: A Natural Language Processing Perspective*. Cambridge, U.K.: Cambridge University Press.
- Jackendoff, R. 1990. *Semantic Structures*. Cambridge, MA: MIT Press.
- Jackendoff, R. 2002. *Foundations of Language: Brain, Meaning, Grammar, Evolution*. Oxford, U.K.: Oxford University Press.
- Johnson, M. 1987. *The Body in the Mind: The Bodily Basis of Meaning, Imagination, and Reason*. Chicago, IL: University of Chicago Press.
- Kamp, H. 1981. A theory of truth and semantic representation. In *Formal Methods in the Study of Language*, J. Groenendijk, T. Janssen, and M. Stokhof (eds.), pp. 277–322. Amsterdam, the Netherlands: Mathematical Centre. [Reprinted in Portner and Partee (eds.) (2002), pp. 189–222; Reprinted in Gutiérrez-Rexach (ed.) (2003) Vol. VI, pp. 158–196].
- Kamp, H. and U. Reyle. 1993. *From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Dordrecht, the Netherlands: Kluwer.
- Kamp, H. and U. Reyle. 1996. A calculus for first order discourse representation structures. *Journal of Logic, Language, and Information* 5(3/4):297–348.
- Karttunen, L. 1976. Discourse referents. In *Syntax and Semantics 7*, J. McCawley (ed.), pp. 363–385. New York, NY: Academic Press.
- Lakoff, G. 1987. *Women, Fire and Dangerous Things. What Categories Reveal about the Mind*. Chicago, IL: Chicago University Press.
- van Lambalgen, M. and F. Hamm. 2005. *The Proper Treatment of Events*. Malden, MA: Blackwell.
- Langacker, R. W. 1987. *Foundations of Cognitive Grammar, Vol. 1: Theoretical Prerequisites*. Stanford, CA: Stanford University Press.
- Langacker, R. W. 1990. *Concept, Image, and Symbol. The Cognitive Basis of Grammar*. Berlin, Germany: Mouton de Gruyter.
- Langacker, R. W. 1991. *Foundations of Cognitive Grammar, Vol. 2: Descriptive Application*. Stanford, CA: Stanford University Press.
- Lappin, S. (ed.). 1997. *The Handbook of Contemporary Semantic Theory*. Oxford, U.K.: Blackwell.
- Lyons, J. 1977. *Semantics*. Cambridge, U.K.: Cambridge University Press.
- Majid, A. and M. Bowerman (eds.). 2007. “Cutting and breaking” events — A cross-linguistic perspective (Special Issue). *Cognitive Linguistics* 18(2).
- Mark, D. M. and A. G. Turk. 2003. Landscape categories in Yindjibarndi: Ontology, environment, and language. In *Spatial Information Theory: Foundations of Geographic Information Science*.

Proceedings of the International Conference COSIT 2003, Kartause Ittingen, Switzerland, September 24–28, 2003, W. Kuhn, M. Worboys, and S. Timpf (eds.). *Lecture Notes in Computer Science* 2825:28–45. Berlin, Germany: Springer-Verlag.

Mark, D. M. and A. G. Turk. 2004. Ethnophysiography and the ontology of landscape. In *Proceedings of GIScience*, Adelphi, MD, October 20–23, 2004, pp. 152–155.

Mark, D. M., A. G. Turk, and D. Stea. 2007. Progress on Yindjibarndi Ethnophysiography. In *Spatial Information Theory. Proceedings of the International Conference COSIT 2007*, Melbourne, Australia,

- September 19–23, 2007, S. Winter, M. Duckham, and L. Kulik (eds.). *Lecture Notes in Computer Science* 4736:1–19. Berlin, Germany: Springer-Verlag.
- Martin, P. 2002. Knowledge representation in CGLF, CGIF, KIF, Frame-CG, and Formalized English. In *Conceptual Structures: Integration and Interfaces. Proceedings of the 10th International Conference on Conceptual Structures, ICSS 2002*, Borovets, Bulgaria, July 15–19, 2002, U. Priss, D. Corbett, and G. Angelova (eds.). *Lecture Notes in Artificial Intelligence* 2393:77–91. Berlin, Germany: Springer-Verlag.
- Mihatsch, W. 2007. Taxonomic and meronomic superordinates with nominal coding. In *Ontolinguistics: How Ontological Status Shapes the Linguistic Coding of Concepts*, A. C. Schalley and D. Zaefferer (eds.), pp. 359–377. Berlin, Germany: Mouton de Gruyter.
- Montague, R. 1973. The proper treatment of quantification in ordinary English. In *Approaches to Natural Language. Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*, J. Hintikka, J. M. E. Moravcsik, and P. Suppes (eds.), pp. 221–242. Dordrecht, the Netherlands: Reidel. [Reprinted in Montague (1974), pp. 247–270; Reprinted in Portner and Partee (eds.) (2002), pp. 17–34; Reprinted in Gutiérrez-Rexach (ed.) (2003) Vol. I, pp. 225–244].
- Montague, R. 1974. *Formal Philosophy: Selected Papers of Richard Montague*, ed. and with an intr. by R. H. Thomason. New Haven, CT: Yale University Press.
- Morato, J., M. Á. Marzal, J. Lloréns, and J. Moreiro. 2004. WordNet applications. In *GWC 2004: Proceedings of the Second International WordNet Conference*, P. Sojka, K. Pala, P. Smrž, C. Fellbaum, and P. Vossen (eds.), pp. 270–278. Brno, Czech Republic: Masaryk University.
- Nerrière, J.-P. 2004. *Don't Speak English, Parlez Globish*. Paris, France: Eyrolles.
- Nickles, M., A. Pease, A. C. Schalley, and D. Zaefferer. 2007. Ontologies across disciplines. In *Ontolinguistics: How Ontological Status Shapes the Linguistic Coding of Concepts*, A. C. Schalley and D. Zaefferer (eds.), pp. 23–67. Berlin, Germany: Mouton de Gruyter.
- Nirenburg, S. and V. Raskin. 2004. *Ontological Semantics*. Cambridge, MA: MIT Press.
- Object Management Group (1997–2009). *Unified Modeling Language. UML Resource Page*. <http://www.uml.org>
- Oosterom, P. van and S. Zlatanova (eds.). 2008. *Creating Spatial Information Infrastructures: Towards the Spatial Semantic Web*. Boca Raton, FL: CRC Press.
- Peeters, B. (ed.). 2006. *Semantic Primes and Universal Grammar: Empirical Evidence from the Romance Languages*. Amsterdam, the Netherlands: John Benjamins.
- Poesio, M. 2000. Semantic analysis. In *Handbook of Natural Language Processing*, R. Dale, H. Moisl, and H. Somers (eds.), pp. 93–122. New York, NY/Basel, Switzerland: Marcel Dekker.
- Pollard, C. and I. A. Sag 1994. *Head-Driven Phrase Structure Grammar*. Chicago, IL/Stanford, CA: University of Chicago Press and CSLI.
- Portner, P. and B. H. Partee (eds.). 2002. *Formal Semantics: The Essential Readings*. Oxford, U.K.: Blackwell.
- Pustejovsky, J. 1991a. The generative lexicon. *Computational Linguistics* 17(4):409–441.
- Pustejovsky, J. 1991b. The syntax of event structure. *Cognition* 41:47–81.
- Pustejovsky, J. 1995. *The Generative Lexicon*. Cambridge, MA: MIT Press.

- Pustejovsky, J. 2001. Generativity and explanation in semantics: A reply to Fodor and Lepore. In *The Language of Word Meaning*, P. Bouillon and F. Busa (eds.), pp. 51–74. Cambridge, U.K.: Cambridge University Press.
- Rosch, E. H. 1973. On the internal structures of perceptual and semantic categories. In *Cognitive Development and the Acquisition of Language*, T. E. Moore (ed.), pp. 111–144. New York, NY: Academic Press.
- Rosch, E., C. B. Mervis, W. D. Gray, D. M. Johnson, and P. Boynes-Braem. 1976. Basic objects in natural categories. *Cognitive Psychology* 8:382–439.
- Ruppenhofer, J., M. Ellsworth, M. R. L. Petruck, C. R. Johnson, and J. Scheffczyk. 2006. *FrameNet II: Extended Theory and Practice*. <http://framenet.icsi.berkeley.edu./book/book.pdf>

- Russell, J. A. 1991. Culture and the categorization of emotion. *Psychological Bulletin* 110:426–450.
- Russell, J. A. and M. Yik. 1996. Emotion among the Chinese. In *Handbook of Chinese Psychology*, M. H. Bond (ed.), pp. 166–188. Hong Kong, China: Oxford University Press.
- Saurer, W. 1993. A natural deduction system for discourse representation theory. *Journal of Philosophical Logic* 22(3):249–302.
- Schalley, A. C. 2004a. *Cognitive Modeling and Verbal Semantics. A Representational Framework Based on UML*. Berlin, Germany/New York, NY: Mouton de Gruyter.
- Schalley, A. C. 2004b. Representing verbal semantics with diagrams. An adaptation of the UML for lexical semantics. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, Geneva, Switzerland, Vol. II, pp. 785–791. Geneva, Switzerland: Association for Computational Linguistics.
- Schalley, A. C. and D. Khlentzos (eds.). 2007. *Mental States. Volume 2: Language and Cognitive Structure*. Amsterdam, the Netherlands: John Benjamins.
- Schalley, A. C. and D. Zaefferer (eds.). 2007. *Ontolinguistics: How Ontological Status Shapes the Linguistic Coding of Concepts*. Berlin, Germany: Mouton de Gruyter.
- Scharl, A. 2007. Towards the geospatial web: Media platforms for managing geotagged knowledge repositories. In *The Geospatial Web—How Geo-Browsers, Social Software and the Web 2.0 are Shaping the Network Society*, A. Scharl and K. Tochtermann (eds.), pp. 3–14. London, U.K.: Springer-Verlag.
- Sowa, J. 2000. *Knowledge Representation. Logical, Philosophical, and Computational Foundations*. Pacific Grove, CA: Brooks/Cole.
- Stock, K. 2008. Determining semantic similarity of behaviour using natural semantic metalanguage to match user objectives to available web services. *Transactions in GIS* 12(6):733–755.
- Talmy, L. 2000. *Toward a Cognitive Semantics*. 2 Vols. Cambridge, MA: MIT Press.
- Van Valin, R. D. 1999. Generalised semantic roles and the syntax-semantics interface. In *Empirical Issues in Formal Syntax and Semantics 2*, F. Corblin, C. Dobrovie-Sorin, and J.-M. Marandin (eds.), pp. 373–389. The Hague, the Netherlands: Thesus. Available at http://linguistics.buffalo.edu/people/faculty/vanvalin/rrg/vanvalin_papers/gensemroles.pdf
- Van Valin, R. D. 2004. Semantic macroroles in Role and Reference Grammar. In *Semantische Rollen*, R. Kailuweit and M. Hummel (eds.), pp. 62–82. Tübingen, Germany: Narr.
- Van Valin, R. D., and R. J. LaPolla. 1997. *Syntax: Structure, Meaning and Function*. Cambridge, U.K.: Cambridge University Press.
- Veres, C. and J. Sampson. 2005. Ontology and taxonomy: Why “is-a” still isn’t just “is-a.” In *Proceedings of the 2005 International Conference on e-Business, Enterprise Information Systems, e-Government, and Outsourcing*, Las Vegas, NV, June 20–23, 2005, H. R. Arabnia (ed.), pp. 174–186. Las Vegas, NV: CSREA Press.
- Vossen, P. 2001. Condensed meaning in EuroWordNet. In *The Language of Word Meaning*, P. Bouillon and F. Busa (eds.), pp. 363–383. Cambridge, U.K.: Cambridge University Press.
- Vossen, P. (ed.). 1998. *EuroWordNet: A Multilingual Database With Lexical Semantic Networks*. Dordrecht, the Netherlands: Kluwer. [Reprinted from *Computers and the Humanities* 32(2/3)]
- Wanner, L. (ed.). 1996. *Lexical Functions in Lexicography and Natural Language Processing*. Amsterdam, the Netherlands: John Benjamins.

- Wanner, L. (ed.). 2007. *Selected Lexical and Grammatical Issues in Meaning-Text Theory: In Honour of Igor Mel'čuk*. Amsterdam, the Netherlands: John Benjamins.
- Wierzbicka, A. 1972. *Semantic Primitives*. Frankfurt, Germany: Athenäum.
- Wierzbicka, A. 1982. Why can you *have a drink* when you can't **have an eat*? *Language* 58(4):753–799.
- Wierzbicka, A. 1985. *Lexicography and Conceptual Analysis*. Ann Arbor, MI: Karoma.
- Wierzbicka, A. 1988. *The Semantics of Grammar*. Amsterdam, the Netherlands: John Benjamins.
- Wierzbicka, A. 1991. Semantic complexity: Conceptual primitives and the principle of substitutability. *Theoretical Linguistics* 17(1/2/3):75–97.

- Wierzbicka, A. 1992. *Semantics, Culture and Cognition: Universal Human Concepts in Culture-Specific Configurations*. Oxford, U.K.: Oxford University Press.
- Wierzbicka, A. 1996. *Semantics: Primes and Universals*. Oxford, U.K.: Oxford University Press.
- Wierzbicka, A. 1997. *Understanding Cultures Through Their Key Words: English, Russian, Polish, German and Japanese*. Oxford, U.K.: Oxford University Press.
- Wierzbicka, A. 1999. *Emotions across Languages and Cultures*. Cambridge, U.K.: Cambridge University Press.
- Wierzbicka, A. 2004. 'Happiness' in cross-linguistic and cross-cultural perspective. *Daedalus* Spring 2004, 133(2):34–43.
- Wierzbicka, A. 2006. *English: Meaning and Culture*. New York, NY: Oxford University Press.
- Wierzbicka, A. 2009a. Overcoming anglocentrism in emotion research. *Emotion Review* 1:24–30.
- Wierzbicka, A. 2009b. *Experience, Evidence, and Sense: The Hidden Cultural Legacy of English*. New York, NY: Oxford University Press.
- Wisniewski, E. J., M. Imai, and L. Casey. 1996. On the equivalence of superordinate concepts. *Cognition* 60:269–298.
- Wunderlich, D. 1996. Models of lexical decomposition. In *Lexical Structures and Language Use. Proceedings of the International Conference on Lexicology and Lexical Semantics*, Vol. 1, *Plenary Lectures and Session Papers*, E. Weigand and F. Hundsnurscher (eds.). pp. 169–183. Tübingen, Germany: Niemeyer.
- Wunderlich, D. 1997. Cause and the structure of verbs. *Linguistic Inquiry* 28(1):27–68.
- Ye, Z. 2001. An inquiry into "sadness" in Chinese. In *Emotions in Crosslinguistic Perspective*, J. Harkins and A. Wierzbicka (eds.), pp. 359–404. Berlin, Germany: Mouton de Gruyter.
- Ye, Z. 2004. Chinese categorization of interpersonal relationships and the cultural logic of Chinese social categories: An indigenous perspective. *Intercultural Pragmatics* 1(2):211–230.
- Zaefferer, D. 2002. Polysemy, polyvalence, and linking mismatches. The concept of RAIN and its codings in English, German, Italian, and Spanish. *DELTA—Documentação de Estudos em Lingüística Teórica e Aplicada* 18(spe):27–56.

6

Natural Language Generation

6.1	Introduction	121
	Generation Compared to Comprehension · Computers Are Dumb · The Problem of the Source	
6.2	Examples of Generated Texts: From Complex to Simple and Back Again	124
	Complex · Simple · Today	
6.3	The Components of a Generator	126
	Components and Levels of Representation	
6.4	Approaches to Text Planning	129
	The Function of the Speaker · Desiderata for Text Planning · Pushing vs. Pulling · Planning by Progressive Refinement of the Speaker's Message · Planning Using Rhetorical Operators · Text Schemas	
6.5	The Linguistic Component	134
	Surface Realization Components · Relationship to Linguistic Theory · Chunk Size · Assembling vs. Navigating · Systemic Grammars · Functional Unification Grammars	
6.6	The Cutting Edge	138
	Story Generation · Personality-Sensitive Generation	
6.7	Conclusions	140
	References	141

David D. McDonald
BBN Technologies

6.1 Introduction

Natural language generation (NLG) is the process by which thought is rendered into language. It has been studied by philosophers, neurologists, psycholinguists, child psychologists, and linguists. Here, we examine what generation is to those who look at it from a computational perspective: people in the fields of artificial intelligence and computational linguistics.

From this viewpoint, the ‘generator’ — the equivalent of a person with something to say — is a computer program. Its work begins with the initial intention to communicate, and then on to determining the content of what will be said, selecting the wording and rhetorical organization and fitting it to a grammar, through to formatting the words of a written text or establishing the prosody of speech. Today, what a generator produces can range from a single word or phrase given in answer to a question or as label on a diagram, through multi-sentence remarks and questions within a dialog, and on to multipage explanations and beyond depending on the capacity and goals of the program it is working for — the machine ‘speaker’

with something to say — and the demands and particulars of the context.

Modulo a number of caveats discussed later, the process of generation is usually divided into three parts, often implemented as three separate programs: (1) identifying the goals of the utterance, (2) planning how the goals may be achieved by evaluating the situation and available communicative resources, and (3) realizing the plans as a text.

Generation has been part of computational linguistics for as long as the field has existed, though it only became a substantial subfield in the 1980s. It appeared first in the 1950s as a minor aspect of machine translation. In the 1960s, random sentence generators were developed, often for use as grammar checkers. The 1970s saw the first cases of dynamically generating the motivated utterances of an artificial speaker: composing answers to questions put to database query programs and providing simple explanations for expert systems. That period also saw the first theoretically important generation systems. These systems reasoned, introspected, appreciated the conventions of discourse, and used sophisticated models of grammar. The texts they produced, while small in number, remain among the most fluent in the literature. By the beginning of the 1980s, generation had emerged as a field of its own, with unique concerns and issues.

6.1.1 Generation Compared to Comprehension

To understand those issues, it will be useful to compare generation with its far more studied and sophisticated cousin, natural language comprehension. Even after 40 years, generation is often misunderstood as a simple variation on comprehension—a tendency that should be dispelled. Generation must be seen as a problem of construction and planning, not analysis.

As a process, generation has its own basis of organization, a fact that follows directly from intrinsic differences in information flow. The processing in language comprehension typically follows the traditional stages of a linguistic analysis: phonology, morphology, syntax, semantics, pragmatics/discourse; moving gradually from the text to the intentions behind it. In comprehension, the ‘known’ is the wording of the text (and possibly its intonation). From the wording, the comprehension process constructs and deduces the propositional content conveyed by the text and the probable intentions of the speaker in producing it. The primary process involves scanning the words of the text in sequence, during which the form of the text gradually unfolds. The need to scan imposes a methodology based on the management of multiple hypotheses and predictions that feed a representation that must be expanded dynamically. Major problems are caused by ambiguity (one form can convey a range of alternative meanings), and by under-specification (the audience gets more information from inferences based on the situation than is conveyed by the actual text). In addition, mismatches in the speaker’s and audience’s model of the situation (and especially of each other) lead to unintended inferences.

Generation has the opposite information flow: from intentions to text, content to form. What is already known and what must be discovered is quite different from comprehension, and this has many implications. The known is the generator’s awareness of its speaker’s intentions and mood, its plans, and the content and structure of any text the generator has already produced. Coupled with a model of the audience, the situation, and the discourse, this information provides the basis for making choices among the alternative wordings and constructions that the language provides—the primary effort in deliberately constructing a text.

Most generation systems do produce texts sequentially from left to right, but only after having made decisions top-down for the content and form of the text as a whole. Ambiguity in a generator’s knowledge is not possible (indeed one of the problems is to notice that an ambiguity has inadvertently been introduced into the text). Rather than under-specification, a generator’s problem is how to choose how to signal its intended inferences from an oversupply of possibilities along with that what information should be omitted and what must be included.

With its opposite flow of information, it would be reasonable to assume that the generation process can be organized like the comprehension process but with the stages in opposite order, and to a certain

extent this is true: pragmatics (goal selection) typically precedes consideration of discourse structure and

coherence, which usually precede semantic matters such as the fitting of concepts to words. In turn, the syntactic context of a word must be fixed before the precise morphological and suprasegmental form it should take can be known. However, we should avoid taking this as the driving force in a generator's design, since to emphasize the ordering of representational levels derived from theoretical linguistics would be to miss generation's special character, namely, that generation is above all a planning process. Generation entails realizing goals in the presence of constraints and dealing with the implications of limitations on resources.*

This being said, the consensus among people who have studied both is that generation is the more difficult of the two. What a person needs to know in order to develop a computer program that produces fluent text is either trivial (the text is entered directly into the code, perhaps with some parameters, and produced as is — virtually every commercial program in wide use that produces text uses this 'template' method) or else it is quite difficult because one has to work out a significant number of techniques and facts about language that other areas of language research have never considered. It is probably no accident that for most of its history advances in NLG have only come through the work of graduate students on their PhD theses.

This also goes a long way toward explaining why so little work has been done on generation as compared with comprehension. At a general meeting, papers on parsing will outnumber those on generation by easily five to one or more. Instead, most work on generation is reported at the international workshops on generation, which have been held nearly every year since 1983.

6.1.2 Computers Are Dumb

Two other difficulties with doing research on generation should be cited before moving on. One just alluded to is the relative stupidity of computer programs, and with it the lack of any practical need for natural language generation as those in the field view it — templates will do just fine. We have seen this with the popular success of programs such as Alice (Wallace), perhaps the best known of the chatterbots that use a large set of stimulus-response rules and clever script writing to simulate an intelligent agent while in fact having virtually no comprehension of what they are saying or what is said to them — a line of work that goes back to Weizenbaum's Eliza (1966).

People who study generation tend more to be scientists than engineers and are trying to understand the human capacity to use language — with all its subtleties of nuance, and the complexity, even arbitrariness, of its motivations. Computers, on the other hand, do not think very subtle thoughts. The authors of their programs, even artificial intelligence programs, inevitably leave out the rationales and goals behind the instructions for their behavior, and with very few exceptions,[†] computer programs do not have any emotional or even rhetorical attitudes toward the people who are using them. Without the richness of information, perspective, and intention that humans bring to what they say, computers have no basis for making the decisions that go into natural utterances. It does not make sense to include a natural language generator in one's system if there is nothing for it to do.

6.1.3 The Problem of the Source

The other difficulty is ultimately more serious and is in large part responsible for the relative lack of sophistication in the field as compared with other language processing disciplines. This is the problem of

* Examples of limited resources include the expressive capacity of the syntactic and lexical devices a given language happens to have, or the limited space available in a sentence or a figure title given the prose style that has been chosen.

[†] The exceptions are programs deliberately written to entertain or interact with people. The animated characters developed

at Zoesis Inc. are a prime example (Loyall et al. 2004) as is the work of Mateas and Stern (2002) on Facade that uses the same technology. While less emotionally grounded, the synthetic characters developed at the USC Institute for Creative Technologies such as their Iraqi Tutor or Mission Rehearsal Exercise are also rich enough to know what to say and why; see, e.g., Traum et al. (2007) or Swartout et al. (2006).

the source. We know virtually nothing about what a generation system should start from if it is to speak as well as people do.

Even when approached as a problem in artificial intelligence rather than human psycholinguistics, this lack of a definitive and well-understood starting point remains a problem, since unlike the situation with automated chess players or the expert systems that control factories, we know next to nothing about how our only examples of effective natural language generators—people—go about the business of producing an utterance.

In language comprehension the source is obvious; we all know what a written text or an acoustic signal is. In generation, the source is a ‘state of mind’ inside a speaker with ‘intentions’ acting in a ‘situation’—all terms of art with very slippery meanings. Studying it from a computational perspective, as we are here, we presume that this state of mind has a representation, but there are dozens of formal (consistently implementable) representations used within the Artificial Intelligence (AI) community that have (what we assume is) the necessary expressive power, with no a priori reason to expect one to be better than another as the mental source of an utterance. Worse yet is the lack of consistency between research groups in their choice of primitive terms and relations—does the representation of a meal bottom-out with ‘eat,’ or must that notion necessarily be expanded into a manner, a result, and a time period, with ‘eat’ just a runtime abstraction.

The lack of a consistent answer to the question of the generator’s source has been at the heart of the problem of how to make research on generation intelligible and engaging for the rest of the computational linguistics community, and it has complicated efforts to evaluate alternative treatments even for people in the field. As a result, the ever-increasing effort at comparative evaluation has focused on isolated subproblems such as the generation of referring expressions.

6.2 Examples of Generated Texts: From Complex to Simple and Back Again

If we look at the development of natural language generation in terms of the sorts of texts different systems have produced, we encounter something of a paradox. As the field advanced, the texts got simpler. Only in the last decade have most generation systems begun to produce texts with the sophistication and fluency that was present in the systems of the early 1970s.

6.2.1 Complex

One dramatic example that developed during the earliest period is John Clippinger’s program Erma (1977), which modeled an actual psychoanalytic patient talking to her therapist. It emulated one paragraph of speech by the patient excerpted from extensive transcripts of her conversations. The effort was joint work by Clippinger in his 1974 PhD thesis and Richard Brown in his Bachelor’s thesis (1974).

The paragraph was the result of a computationally complex model of the patient’s thought processes: from the first identification of a goal, through planning, criticism, and replanning of how to express it, and finally linguistic realization. Clippinger and Brown’s program had a multiprocessing capability—it could continue to think and plan while talking. This allowed them to develop a model of ‘restart’ phenomena in generation, including the motivation behind fillers like “*uh*” or dubitatives like “*you know*.” Text segments shown below in parenthesis are what Erma was planning to say before it cut itself off and restarted. In other respects, this is an actual paragraph from a transcript of the patient reproduced in every detail, but from a first principles model of thought and generation.

You know for some reason I just thought about the bill and payment again. (You shouldn't give me a bill.) <Uh> I was thinking that I (shouldn't be given a bill) of asking you whether it wouldn't be all

right for you not to give me a bill. That is, I usually by (the end of the month know the amount of the bill), well, I immediately thought of the objections to this, but my idea was that I would simply count up the number of hours and give you a check at the end of the month.

There has yet to be another program in the literature that can even begin to approach the human-like quality of this text.* On the other hand Erma only ever produced that one text and some parameter-driven variations, and neither Brown's multilevel, resumable, interrupt-driven computational architecture nor Clippinger's rich set of thinking, critiquing, and linguistic modules were ever followed up by other people.

6.2.2 Simple

By the end of the decade of the 1970s, generation began to be recognized as a field with shared assumptions and not just the work of scattered individuals. It also began to attract the attention of the research-funding community, a mixed blessing perhaps, since while the additional resources now allowed work on generation to be pursued by research teams instead of isolated graduate students, the need to conform to the expectations of other groups — particularly in the choice of source representation and conceptual vocabulary — substantially limits the creative options. Probably as a direct result, the focus of the work during the 1980s moved within the generator, and the representations and architecture of the speaker became a black box behind an impenetrable wall.

Nevertheless, the greatly increased number of people working in the field led to many important developments. If the texts that the various groups' systems produced were not of the highest quality, this was offset by increased systematicity in the techniques in use, and a markedly greater understanding of some of the specific issues in generation. Among these were the following:

- The implications of separating the processing of a generator into distinct modules and levels of representation, especially in regard to which operations (lexical choice, linear ordering, and such) took place at which level
- The use of pronouns and other forms of subsequent reference
- The possibilities and techniques for 'aggregating' minimal propositions to form syntactically complex texts
- The relationship between how lexical choice is done and the choice of representation used in the source

Here is an example of text produced by systems developed in the late 1980s — a generator that is not at least this fluent today would be well behind the state of the art. This is from Marie Meteer's Spokesman system (1992), and set in a military domain; the text shown here is an excerpt from a page-long generated operations order (OPORD). Notice the use of simple formatting elements.

2. MISSION

10th Corps defend in assigned sector to defeat the 8th Combined Arms Army.

3. EXECUTION

a. 52d Mechanized Division

(1) Conduct covering force operations along avenues B and C to defeat the lead regiments of the first tactical echelon in the CFA in assigned sector.

A text like this will never win any prizes for literature, but unlike its hand-crafted predecessors of the 1970s, it can be produced mechanically from any comparable input without any human intervention or fine tuning.

* One notable early exception was in Richard Gabriel's thesis (1981, 1986) where he seamlessly wove three machine generated paragraphs describing a procedure that were indistinguishable from the rest. Today we also see mundane report and web pages that are mixtures

of generations from first principles (from a semantic model) and frozen templates selected by people that are hard tell from the 'real thing.'

The source OPOD for this text was a battle order data structure that was automatically constructed by a simulation system that was part of SIMNET (Cosby 1999) that fought virtual battles in detail against human troops in tank simulators — an excellent source of material for a generator to work with.

6.2.3 Today

Now, as we near the end of the first decade of the twenty-first century, we have reached a point where a well designed and linguistically sophisticated system can achieve the fluency of the special-purpose systems of the 1970s, but will operate on a better understood theoretical base. As an example of this, consider Jacques Robin's *Streak* (1993, 1996). It operates within a sublanguage, in this case the language of sports: it writes short summaries of basketball games. Like all news reporting, this genre is characterized by information-dense, syntactically rich summary texts, texts that remain challenging to the best of systems. With *Streak*, Robin has appreciated the extensive references to historical information in these texts, and his experience has important implications for how to approach the production of summaries of all sorts.

Technically, *Streak* is a system based on revision. It begins by producing a representation of the simple facts that will provide anchors for later extensions. Here is an example of what it could start from (Robin 1996: 206).

Dallas, TX—Charles Barkley scored 42 points Sunday as the Phoenix Suns defeated the Dallas Mavericks 123–97.

This initial text is then modified as salient historical or ancillary information about this game and the players' past records is considered. Here is the final form.

Dallas, TX—Charles Barkley tied a season high with 42 points and Danny Ainge came off the bench to add 21 Sunday as the Phoenix Suns handed the Dallas Mavericks their league worst 13th straight home defeat 123–97.

Notice what has happened. Initial forms have been progressively replaced with phrases that carry more information: “scored *N* points” has become “tied a season high with *N* points.” Syntactic formulations have been changed (“defeat *X*” has become “hand *X(a)* defeat”), where the new choice is able to carry information that the original could not (the noun form of “defeat” can be modified by “their league worst” and “*N*th straight home”). This is sophisticated linguistic reasoning that has been matched by only a few earlier systems.

Given the rich architectures available in generation systems today, the production of detailed, if mundane, information derived directly from an application program has become almost a cookbook operation in the sense that practitioners of the art can readily engineer a system with these abilities in a relatively short period of time. Much of what makes these modern generators effective is that they are applied to very specific domains, domains where the corpus of text can be described as belonging to a ‘sublanguage’ (see, e.g., Kittredge and Lehrberger 1982). That is to say, they restrict themselves to a specialized area of discourse with a very focused audience and stipulated content, thereby reducing the options for word choice and syntactic style to a manageable set. In particular, museum exhibits have been a very profitable area for NLG because they provide a natural setting for tailoring text to appreciate what exhibits people have already heard about. The ILEX system, for example, focused on ordering issues and the dynamic generation of text in Web pages (e.g., O'Donnell et al. 2001 or Dale et al. 1998).

6.3 The Components of a Generator

To produce a text in the computational paradigm, there has to be a program with something to say—we can call this ‘the application’ or ‘the speaker.’ And there must be a program with the competence to

render the application's intentions into fluent prose appropriate to the situation — what we will call 'the generator' — which is the natural language generation system proper.

Given that the task is to engineer the production of text or speech for a purpose — emulating what people do and/or making it available to machines — then both of these components, the speaker and the generator, are necessary. Studying the language side of the process without anchoring the work with respect to the conceptual models and intentional structures of an application may be appropriate for theoretical linguistics or the study of grammar algorithms, but not for language generation. Indeed, some of the most exciting work comes from projects where the generator is only a small part.*

As described earlier, the very earliest work on sophisticated language production interleaved the functions of the speaker and the generator into a single system. Today, there will invariably be three or four components (if not a dozen) dividing the work amongst themselves according to a myriad of different criteria. We will discuss the philosophies governing these criteria later.

6.3.1 Components and Levels of Representation

Given the point of view we adopt in this chapter, we will say that generation starts in the mind of the speaker (the execution states of the computer program) as it acts upon an intention to say something — to achieve some goal through the use of language: to express feelings, to gossip, to assemble a pamphlet on how to stop smoking (Reiter et al. 2003).

6.3.1.1 Tasks

Regardless of the approach taken, generation proper involves at least four tasks.

- a. Information must be *selected* for inclusion in the utterance.
Depending on how this information is reified into representational units (a property of the speaker's mental model), parts of the units may have to be omitted, other units added in by default, and perspectives taken on the units to reflect the speaker's attitude toward them.
- b. The information must be given a *textual organization*.
It must be ordered, both sequentially and in terms of linguistic relations such as modification or subordination. The coherence relationships among the units of the information must be reflected in this organization so that the reasons why the information was included will be apparent to the audience.
- c. *Linguistic resources* must be chosen to support the information's realization.
Ultimately these resources will come down to choices of particular words, idioms, syntactic constructions, productive morphological variations, etc., but the form they take at the first moment that they are associated with the selected information will vary greatly between approaches. (Note that to choose a resource is not ipso facto to simultaneously deploy it in its final form — a fact that is not always appreciated.)
- d. The selected and organized resources must be *realized* as an actual text and written out or spoken.
This stage can itself involve several levels of representation and interleaved processes.

6.3.1.2 Coarse Components

These four tasks are usually divided among three components as listed below. The first two are often spoken of as deciding 'what to say,' the third deciding 'how to say it.'

* See for example the integration of generation into dynamically produced movies that act as tour guides in the Peach system (Callaway et al. 2005, Stock and Zancanaro 2007).

1. The application program or 'speaker.'

It does the thinking and maintains a model of the situation. Its goals are what initiate the process, and it is its representation of concepts and the world that supplies the source on which the other components operate.

2. A text planner.

It selects (or receives) units from the application and organizes them to create a structure for the utterance as a text by employing some knowledge of rhetoric. It appreciates the conventions for signaling information flow in a linguistic medium: what information is new to the interlocutors, what is old; what items are in focus; and whether there has been a shift in topic.

3. A linguistic component.

It realizes the planner's output as an utterance. In its traditional form during the 1970s and early 1980s it supplied all of the grammatical knowledge used in the generator. Today this knowledge is likely to be more evenly distributed throughout the system. This component's task is to adapt (and possibly to select) linguistic forms to fit their grammatical contexts and to orchestrate their composition. This process leads, possibly incrementally, to a surface structure for the utterance, which is then read out to produce the grammatically and morphologically appropriate wording for the utterance.

How these roughly drawn components interact is a matter of considerable debate and no little amount of confusion, as no two research groups are likely to agree on precisely what kinds of knowledge or processing appear in a given component or where its boundaries should lie. There have been attempts to standardize the process, most notably the RAGS project (see, e.g., Cahill et al. 1999), but to date they have failed to gain any traction.

One camp, making an analogy to the apparent abilities of people, holds that the process is monotonic and indelible. A completely opposite camp extensively revises its (abstract) draft texts. Some groups organize the components as a pipeline; others use blackboards. Nothing conclusive about the relative merits of these alternatives can be said today. We continue to be in a period where the best advice is to let a 1000 flowers bloom.

6.3.1.3 Representational Levels

There are necessarily one or more intermediate levels between the source and the text simply because the production of an utterance is a serial process extended in time. Most decisions will influence several parts of the utterance at once, and consequently cannot possibly be acted upon at the moment they are made. Without some representation of the results of these decisions there would be no mechanism for remembering them and utterances would be incoherent.

The consensus favors at least three representational levels, roughly the output of each of the components. In the first or 'earliest' level, the information units of the application that are relevant to the text planner form a *message* level—the source from which the later components operate. Depending on the system, this level can consist of anything from an unorganized heap of minimal propositions or RDF to an elaborate typed structure with annotations about the relevance and purposes of its parts.

All systems include one or more levels of *surface syntactic structure*. These encode the phrase structure of the text and the grammatical relations among its constituents. Morphological specialization of word stems and the introduction of punctuation or capitalization are typically done as this level is read out and the utterance uttered. Common formalisms at this level include systemic networks, tree-adjointing and categorial grammar, and functional unification, though practically every linguistic theory of grammar that has ever been developed has been used for generation at one time or another. Nearly all of today's generation systems express their utterances as writ-

ten texts—characters printed on a computer screen or printed out as a pamphlet—rather than

as speech. Consequently generators seldom include an explicit level of phonological form and intonation.*

In between the message and the surface structure is a level (or levels) of representation at which a system can reason about linguistic options without simultaneously being committed to syntactic details that are irrelevant to the problem at hand. Instead, abstract linguistic structures are combined with generalizations of the concepts in the speaker's domain-specific model and sophisticated concepts from lexical semantics. The level is variously called *text structure*, *deep syntax*, *abstract syntactic structure*, and the like. In some designs, it will employ rhetorical categories such as elaboration or temporal location. Alternatively it may be based on abstract linguistic concepts such as the matrix–adjunct distinction. It is usually organized as trees of constituents with a layout roughly parallel to that of the final text. The leaves of these trees may be direct mappings of units from the application or may be semantic structures specific to that level.

6.4 Approaches to Text Planning

Even though the classic conception of the division of labor in generation between a text planner and a linguistic component – where the latter is the sole repository of the generator's knowledge of language – was probably never really true in practice and is certainly not true today, it remains an effective expository device. In this section, we consider text planning in a relatively pure form, concentrating on the techniques for determining the content of the utterance and its large-scale (supra-sentential) organization.

It is useful in this context to consider a distinction put forward by the psycholinguist Willem Levelt (1989), between 'macro' and 'micro' planning.

- *Macro-planning* refers to the process(es) that choose the speech acts, establish the content, determine how the situation dictates perspectives, and so on.
- *Micro-planning* is a cover term for a group of phenomena: determining the detailed (sentence-internal) organization of the utterance, considering whether to use pronouns, looking at alternative ways to group information into phrases, noting the focus and information structure that must apply, and other such relatively fine-grained tasks. These, along with lexical choice, are precisely the set of tasks that fall into this nebulous middle ground that is motivating so much of today's work.

6.4.1 The Function of the Speaker

From the generator's perspective, the function of the application that it is working for is to set the scene. Since it takes no overtly linguistic actions beyond initiating the process, we are not inclined to think of the application program as a part of the generator proper. Nevertheless, the influence it wields in defining the situation and the semantic model from which the generator works is so strong that it must be designed in concert with the generator if high-quality results are to be achieved. This is the reason why we often speak of the application as the 'speaker,' emphasizing the linguistic influences on its design and its tight integration with the generator.

The speaker establishes what content is potentially relevant. It maintains an attitude toward its audience (as a tutor, reference guide, commentator, executive summarizer, copywriter, etc.). It has a history of past transactions. It is the component with the model of the present state and its physical or conceptual context. The speaker deploys a representation of what it knows, and this implicitly determines the nature and the expressive potential of the 'units' of speaker stuff that the generator works from to produce the

* Again, the exceptions are systems that are specifically designed to talk with people, particularly multi-modal systems that combine speech with gesture. The work by Justine Cassell on Rea (2000) is a prime example. The need to coordinate (animated) gesture with the production of the speech down to the syllable motivates a level representing coordinated action plans.

utterance (the source). We can collectively characterize all of this as the ‘situation’ in which the generation of the utterance takes place, in the sense of Barwise and Perry (1983) (see also Devlin 1991).

In the simplest case, the application consists of just a passive data base of items and propositions, and the situation is a selected subset of those propositions (the ‘relevant data’) that has been selected through some means, often by following the thread of a set of identifiers chosen in response to a question from the user.

In some cases, the situation is a body of raw data and the job of speaker is to make sense of it in linguistically communicable terms before any significant work can be done by the other components. The literature includes several important systems of this sort. Probably the most thoroughly documented is the Ana system developed by Karen Kukich (1986), where the input is a set of time points giving the values of stock indexes and trading volumes during the course of a day.

When the speaker is a commentator, the situation can evolve from moment to moment in actual real time. The SOCCER system (Andre et al. 1988) did commentary for football games that were being displayed on the user’s screen. This led to some interesting problems in how large a chunk of information could reasonably be generated at a time, since too small a chunk would fail to see the larger intentions behind a sequence of individual passes and interceptions, while too large a chunk would take so long to utter that the commentator would fall behind the action.

One of the crucial tasks that must often be performed at the juncture between the application and the generator is enriching the information that the application supplies so that it will use the concepts that a person would expect even if the application had not needed them. We can see an example of this in one of the earliest, and still among the most accomplished generation systems, Anthony Davey’s Proteus (1974).

Proteus played games of tic-tac-toe (noughts and crosses) and provided commentary on the results. Here is an example of what it produced:

The game started with my taking a corner, and you took an adjacent one. I threatened you by taking the middle of the edge opposite that and adjacent to the one which I had just taken but you blocked it and threatened me. I blocked your diagonal and forked you. If you had blocked mine, you would have forked me, but you took the middle of the edge opposite of the corner which I took first and the one which you had just taken and so I won by completing my diagonal.

Proteus began with a list of the moves in the game it had just played. In this sample text, the list was the following. Moves are notated against a numbered grid; square one is the upper left corner. Proteus (P) is playing its author (D).

P:1 D:3 P:4 D:7 P:5 D:6 P:9

One is tempted to call this list of moves the ‘message’ that Proteus’s text-planning component has been tasked by its application (the game player) to render into English—and it is what actually crosses the interface between them—but consider what this putative message leaves out when compared with the ultimate text: where are the concepts of move and countermove or the concept of a fork? The game playing program did not need to think in those terms to carry out its task and performed perfectly well without them, but if they were not in the text we would never for a moment think that the sequence was a game of tic-tac-toe.

Davey was able to get texts of this complexity and naturalness only because he imbued Proteus with a rich conceptual model of the game, and consequently could have it use terms like ‘block’ or ‘threat’ with assurance. Like most instances where exceptionally fluent texts have been produced, Davey was able to get this sort of performance from Proteus because he had the opportunity to develop the thinking part of the system as well its linguistic aspects, and consequently could insure that the speaker supplied rich perspectives and intentions for the generator to work with.

This, unfortunately, is quite a common state of affairs in the relationship between a generator and its speaker. The speaker, as an application program carrying out a task, has a pragmatically complete

but conceptually impoverished model of what it wants to relate to its audience. Concepts that must be explicit in the text are implicit but unrepresented in the application's code and it remains to the generator (Proteus in this case) to make up the difference. Undoubtedly the concepts were present in the mind of the application's human programmer, but leaving them out makes the task easier to program and rarely limits the application's abilities. The problem of most generators is in effect how to convert water to wine, compensating in the generator for limitations in the application (McDonald and Meteer 1988).

6.4.2 Desiderata for Text Planning

The tasks of a text planner are many and varied. They include the following:

- Construing the speaker's situation in realizable terms given the available vocabulary and syntactic resources, an especially important task when the source is raw data. For example, precisely what points of the compass make the wind "*easterly*" (Bourbeau et al. 1990, Reiter et al. 2005)
- Determining the information to include in the utterance and whether it should be stated explicitly or left for inference
- Distributing the information into sentences and giving it an organization that reflects the intended rhetorical force, as well as the appropriate conceptual coherence and textual cohesion given the prior discourse

Since a text has both a literal and a rhetorical content, not to mention reflections of the speaker's affect and emotions, the determination of what the text is to say requires not only a specification of its propositions, statements, references, etc., but also a specification of how these elements are to be related to each other as parts of a single coherent text (what is evidence, what is a digression) and of how they are structured as a presentation to the audience to which the utterance is addressed. This presentation information establishes what is thematic, where the shifts in perspective are, how new information fits within the context established by the text that preceded it, and so on.

How to establish the simple, literal information content of the text is well understood, and a number of different techniques have been extensively discussed in the literature. How to establish the rhetorical content of the text, however, is only beginning to be explored, and in the past was done implicitly or by rote by directly coding it into the program. There have been some experiments in deliberate rhetorical planning, notably by Hovy (1990) and DiMarco and Hirst (1993). The specification and expression of affect is only just beginning to be explored, prompted by the ever increasing use of 'language enabled' synthetic characters in games, for example, Mateas and Stern (2003), and avatar-based man-machine interaction, for example, Piwek et al. (2005) or Streit et al. (2006).

6.4.3 Pushing vs. Pulling

To begin our examination of the major techniques in text planning, we need to consider how the text planner and speaker are connected. The interface between the two is based on one of two logical possibilities: 'pushing' or 'pulling.'

The application can push units of content to the text planner, in effect telling the text planner what to say and leaving it the job of organizing the units into a text with the desired style and rhetorical effect. Alternatively, the application can be passive, taking no part in the generation process, and the text planner will pull units from it. In this scenario, the speaker is assumed to have no intentions and only the simplest ongoing state (often it is a database). All of the work is then done on the generator's side of the fence.

Text planners that pull content from the application establish the organization of the text hand in glove with its content, using models of possible texts and their rhetorical structure as the basis of their actions. Their assessment of the situation determines which model they will use. Speakers that push content to the text planner typically use their own representation of the situation directly as the content source. At

the time of writing, the pull school of thought has dominated new, theoretically interesting work in text

planning, while virtually all practical systems are based on simple push applications or highly stylized, fixed 'schema'-based pull planners.

6.4.4 Planning by Progressive Refinement of the Speaker's Message

This technique — often called 'direct replacement' — is easy to design and implement, and is by far the most mature approach of those we will cover. In its simplest form, it amounts to little more than is done by ordinary database report generators or mail-merge programs when they make substitutions for variables in fixed strings of text. In its sophisticated forms, which invariably incorporate multiple levels of representation and complex abstractions, it has produced some of the most fluent and flexible texts in the field. Three systems discussed earlier did their text planning using progressive refinement: Proteus, Erma, and Spokesman.

Progressive refinement is a push technique. It starts with a data structure already present in the application and then it gradually transforms that data into a text. The semantic coherence of the final text follows from the underlying semantic coherence that is present in the data structure that the application passes to the generator as its message.

The essence of progressive refinement is to have the text planner add additional information on top of the basic skeleton provided by the application. We can see a good example of this in Davey's Proteus system, where in this case the skeleton is the sequence of moves. The ordering of the moves must still be respected in the final text because Proteus is a commentator and the sequence of events described in a text is implicitly understood as reflecting a sequence in the world. Proteus only departs from the ordering when it serves a useful rhetorical purpose, as in the example text where it describes the alternative events that could have occurred if its opponent had made a different move early on.

On top of the skeleton, Proteus looks for opportunities to group moves into compound complex sentences by viewing the sequence of moves in terms of the concepts of tic-tac-toe. For example, it looks for pairs of forced moves (i.e., a blocking move to counter a move that had set up two in a row). It also looks for moves with strategically important consequences (a move creating a fork). For each semantically significant pattern that it knows how to recognize, Proteus has one or more text organization patterns that can express it. For example, the pattern 'high-level action followed by literal statement of the move' might yield "*I threatened you by taking the middle of the edge opposite that.*" Alternatively, Proteus could have used 'literal move followed by its high-level consequence' pattern: "*I took the middle of the opposite edge, threatening you.*"

The choice of realization is left up to a *specialist*, which takes into account as much information as the designer of the system, Davey in this case, knows how to bring to bear. Similarly, a specialist is employed to elaborate on the skeleton when larger scale strategic phenomena occur. In the case of a fork, this prompts the additional rhetorical task of explaining what the other player might have done to avoid the fork.

Proteus' techniques are an example of the standard design for a progressive refinement text planner: start with a skeletal data structure that is a rough approximation of the final text's organization using information provided by the speaker directly from its internal model of the situation. The structure then goes through some number of successive steps of processing and re-representation as its elements are incrementally transformed or mapped to structures that are closer and closer to a surface text, becoming progressively less domain oriented and more linguistic at each step. The Streak system described earlier follows the same design, replacing simple syntactic and lexical forms with more complex ones with a greater capacity to carry content.

Control is usually vested in the structure itself, using what is known as *data-directed control*. Each element of the data is associated with a specialist or an instance of some standard mapping which takes charge of assembling the counterpart of the element within the next layer of representation. The whole process is often organized into a pipeline where processing can be going on at multiple representational levels simultaneously as the text is produced in its natural left to right order as it would unfold if being

spoken by a person.

A systematic problem with progressive refinement follows directly from its strengths, namely, that its input data structure, the source of its content and control structure, is also a straightjacket. While it provides a ready and effective organization for the text, the structure does not provide any vantage point from which to deviate from that organization even if that would be more effective rhetorically. This remains a serious problem with the approach, and is part of the motivation behind the types of text planners we will look at next.

6.4.5 Planning Using Rhetorical Operators

The next text-planning technique that we will look at can be loosely called ‘formal planning using rhetorical operators.’ It is a pull technique that operates over a pool of relevant data that has been identified within the application. The chunks in the pool are typically full propositions—the equivalents of single simple clauses if they were realized in isolation.

This technique assumes that there is no useful organization to the propositions in the pool, or, alternatively, that such organization as is there is orthogonal to the discourse purpose at hand, and should be ignored. Instead, the mechanisms of the text planner look for matches between the items in the relevant data pool and the planner’s abstract patterns, and select and organize the items accordingly.

Three design elements come together in the practice of operator-based text planning, all of which have their roots in work done in the later 1970s:

- The use of formal means–ends reasoning techniques adapted from the robot-action planning literature
- A conception of how communication could be formalized that derives from speech-act theory and specific work done at the University of Toronto
- Theories of the large-scale ‘grammar’ of discourse structure

Means–ends analysis, especially as elaborated in the work by Sacerdoti (1977), is the backbone of the technique. It provides a control structure that does a top–down, hierarchical expansion of goals. Each goal is expanded through the application of a set of operators that instantiate a sequence of subgoals that will achieve it. This process of matching operators to goals terminates in propositions that can directly realize the actions dictated by terminal subgoals. These propositions become the leaves of a tree-structured text plan, with the goals as the nonterminals and the operators as the rules of derivation that give the tree its shape.

6.4.6 Text Schemas

The third text-planning technique we describe is the use of preconstructed, fixed networks that are referred to as ‘schemas’ following the coinage of the person who first articulated this approach, Kathy McKeown (1985). Schemas are a pull technique. They make selections from a pool of relevant data provided by the application according to matches with patterns maintained by the system’s planning knowledge—just like an operator-based planner. The difference is that the choice of (the equivalent of the) operators is fixed rather than actively planned. Means–ends analysis-based systems assemble a sequence of operators dynamically as the planning is underway. A schema-based system comes to the problem with the entire sequence already in hand.

Given that characterization of schemas, it would be easy to see them as nothing more than compiled plans, and one can imagine how such a compiler might work if a means–ends planner were given feedback about the effectiveness of its plans and could choose to reify its particularly effective ones (though no one has ever done this). However, that would miss an important fact about system design that it is often simpler and just as effective to simply write down a plan by rote rather than to attempt to develop a theory of the knowledge of context and communicative effectiveness that would be deployed in the development

of the plan and from that attempt to construct a plan from first principles, which is essentially what the

means–ends approach to text planning does. It is no accident that schema-based systems (and even more so progressive refinement systems) have historically produced longer and more interesting texts than means–ends systems.

Schemas are usually implemented as transition networks, where a unit of information is selected from the pool as each arc is traversed. The major arcs between nodes tend to correspond to chains of common object references between units: cause followed by effect, sequences of events that are traced step by step through time, and so on. Self loops returning back to the same node dictate the addition of attributes to an object, side effects of an action, etc.

The choice of what schema to use is a function of the overall goal. McKeown's original system, for example, dispatched on a three-way choice between defining an object, describing it, or distinguishing it from another type of object. Once the goal is determined, the relevant knowledge pool is separated out from the other parts of the reference knowledge base and the selected schema is applied. Navigation through the schema's network is then a matter of what units or chains of units are actually present in the pool in combination with the tests that the arcs apply.

Given a close fit between the design of the knowledge base and the details of the schema, the resulting texts can be quite good. Such faults as they have are largely the result of weakness in other parts of the generator and not in its content-selection criteria. Experience has shown that basic schemas can be readily abstracted and ported to other domains (McKeown et al. 1990). Schemas do have the weakness when compared to systems with explicit operators and dynamic planning that, when used in interactive dialogs, do not naturally provide the kinds of information that is needed for recognizing the source of problems, which makes it difficult to revise any utterances that are initially not understood (Moore and Swartout 1991, Paris 1991). But, for most of the applications to which generation systems are put, schemas are a simple and easily elaborated technique that is probably the design of choice whenever the needs of the system or nature of the speaker's model make it unreasonable to use progressive refinement.

6.5 The Linguistic Component

In this section, we look at the core issues in the most mature and well defined of all the processes in natural language generation, the application of a grammar to produce a final text from the elements that were decided upon by the earlier processing. This is the one area in the whole field where we find true instances of what software engineers would call properly modular components: bodies of code and representations with well-defined interfaces that can be (and have been) shared between widely varying development groups.

6.5.1 Surface Realization Components

To reflect the narrow scope (but high proficiency) of these components, I refer to them here as surface realization components. 'Surface' (as opposed to deep) because what they are charged with doing is producing the final syntactic and lexical structure of the text – what linguists in the Chomskian tradition would call a surface structure; and 'realization' because what they do never involves planning or decision making: They are in effect carrying out the orders of the earlier components, rendering (realizing) their decisions into the shape that they must take to be proper texts in the target language.

The job of a surface realization component is to take the output of the text planner, render it into a form that can be conformed (in a theory-specific way) to a grammar, and then apply the grammar to arrive at the final text as a syntactically structured sequence of words, which are read out to become the output of the generator as a whole. The relationships between the units of the plan are mapped to syntactic relationships. They are organized into constituents and given a linear ordering. The content words are given grammatically appropriate morphological realizations. Function words ("to," "of," "has," and such)

are added as the grammar dictates.

6.5.2 Relationship to Linguistic Theory

Practically without exception, every modern realization component is an implementation of one of the recognized grammatical formalisms of theoretical linguistics. It is also not an exaggeration to say that virtually every formalism in the alphabet soup of alternatives that is modern linguistics has been used as the basis of some realizer in some project somewhere.

The grammatical theories provide systems of rules, sets of principles, systems of constraints, and, especially, a rich set of representations, which, along with a lexicon (not a trivial part in today's theories), attempt to define the space of possible texts and text fragments in the target natural language. The designers of the realization components devise ways of interpreting these theoretical constructs and notations into effective machinery for constructing texts that conform to these systems.

It is important to note that all grammars are woefully incomplete when it comes to providing accounts (or even descriptions) of the actual range of texts that people produce, and no generator within the present state of the art is going to produce a text that is not explicitly in the competence of the surface grammar it is using. Generation is in a better situation in this respect than comprehension is, however. As a constructive discipline, we at least have the capability of extending our grammars whenever we can determine a motive (by the text planner) and a description (in terms of the grammar) for some new construction. As designers, we can also choose whether to use a construct or not, leaving out everything that is problematic. Comprehension systems on the other hand, must attempt to read the texts they happen to be confronted with and so will inevitably be faced at almost every turn with constructs beyond the competence of their grammar.

6.5.3 Chunk Size

One of the side effects of adopting the grammatical formalisms of the theoretical linguistics community is that every realization component generates a complete sentence at a time, with a few notable exceptions.* Furthermore this choice of 'chunk size' becomes an architectural necessity, not a freely chosen option. As implementations of established theories of grammar, realizers must adopt the same scope over linguistic properties as their parent theories do; anything larger or smaller would be undefined.

The requirement that the input to most surface realization components specify the content of an entire sentence at a time has a profound effect on the planners that must produce these specifications. Given a set of propositions to be communicated, the designer of a planner working in this paradigm is more likely to think in terms of a succession of sentences rather than trying to interleave one proposition within the realization of another (although some of this may be accomplished by aggregation or revision). Such lockstep treatments can be especially confining when higher order propositions are to be communicated. For example, the natural realization of such a proposition might be adding "only" inside the sentence that realizes its argument, yet the full-sentence-at-a-time paradigm makes this exceedingly difficult to appreciate as a possibility let alone carry out.

6.5.4 Assembling vs. Navigating

Grammars, and with them the processing architectures of their realization components, fall into two camps.

* The Mumble-86 realizer (Meteer et al. 1987), when it was used as part of Jeff Conklin's Genaro system (Conklin and McDonald 1982) determined sentence length and composition dynamically according to a "weight" calculated from the character of the constructions it contained and began to populate the next sentence once a threshold parameter had been exceeded. This was

possible because Mumble is based on lexicalized Tree Adjoining Grammar, where the grammar chunks can be as small as a single word.

- The grammar provides a set of relatively small structural elements and constraints on their combination.
- The grammar is a single complex network or descriptive device that defines all the possible output texts in a single abstract structure (or in several structures, one for each major constituent type that it defines: clause, noun phrase, thematic organization, and so on).

When the grammar consists of a set of combinable elements, the task of the realization component is to select from this set and assemble them into a composite representation from which the text is then read out. When the grammar is a single structure, the task is to navigate through the structure, accumulating and refining the basis for the final text along the way and producing it all at once when the process has finished.

Assembly-style systems can produce their texts incrementally by selecting elements from the early parts of the text first, and can thereby have a natural representation of ‘what has already been said’ which is a valuable resource for making decisions about whether to use pronouns and other position-based judgments. Navigation-based systems, because they can see the whole text at once as it emerges, can allow constraints from what will be the later parts of the text to effect realization decisions in earlier parts, but they can find it difficult, even impossible, to make certain position-based judgments.

Among the small-element linguistic formalisms that have been used in generation we have conventional production rule rewrite systems, CCG, Segment Grammar, and Tree Adjoining Grammar (TAG). Among the single-structure formalisms, we have Systemic Grammar and any theory that uses feature structures, for example, HPSG and LFG. We look at two of these in detail because of their influence within the community.

6.5.5 Systemic Grammars

Understanding and representing the context into which the elements of an utterance are fit and the role of the context in their selection is a central part of the development of a grammar. It is especially important when the perspective that the grammarian takes is a functional rather than a structural one – the viewpoint adopted in Systemic Grammar. A structural perspective emphasizes the elements out of which language is built (constituents, lexemes, prosodics, etc.). A functional perspective turns this on its head and asks what is the spectrum of alternative purposes that a text can serve (its ‘communicative potential’). Does it introduce a new object which will be the center of the rest of the discourse? Is it reinforcing that object’s prominence? Is it shifting the focus to something else? Does it question? Enjoin? Persuade? The multitude of goals that a text and its elements can serve provides the basis for a paradigmatic (alternative based) rather than a structural (form based) view of language.

The Systemic Functional Grammar (SFG) view of language originated in the early work of Michael Halliday (1967, 1985) and Halliday and Matthiessen (2004) and has a wide following today. It has always been a natural choice for work in language generation (Davey’s Proteus system was based on it) because much of what a generator must do is to choose among the alternative constructions that the language provides based on the context and the purpose they are to serve – something that a systemic grammar represents directly.

A systemic grammar is written as a specialized kind of decision tree: ‘If this choice is made, then this set of alternatives becomes relevant; if a different choice is made, those alternatives can be ignored, but this other set must now be addressed.’ Sets of (typically disjunctive) alternatives are grouped into ‘systems’ (hence “systemic grammar”) and connected by links from the prior choice(s) that made them relevant to the other systems that they in turn make relevant. These systems are described in a natural and compelling graphic notation of vertical bars listing each system and lines connecting them to other systems. (The Nigel systemic grammar, developed at ISI (Matthiessen 1983), required an entire office wall for its presentation using this notation.)

In a computational treatment of SFG for language generation, each system of alternative choices has an associated decision criteria. In the early stages of development, these criteria are often left to

human intervention so as to exercise the grammar and test the range of constructions it can motivate

(e.g., Fawcett 1981). In the work at ISI, this evolved into what was called ‘inquiry semantics,’ where each system had an associated set of predicates that would test the situation in the speaker’s model and makes its choices accordingly. This makes it in effect a ‘pull’ system for surface realization; something that in other publications has been called *grammar-driven* control as opposed to the *message-driven* approach of a system like Mumble (see McDonald et al. 1987).

As the Nigel grammar grew into the Penman system (Penman Natural Language Group 1989) and gained a wide following in the late 1980s and early 1990s, the control of the decision making and the data that fed it moved from the grammar’s input specification and into the speaker’s knowledge base. At the heart of the knowledge base—the taxonomic lattice that categorizes all of the types of objects that the speaker could talk about and defines their basic properties—an upper structure was developed (Bateman 1997, Bateman et al. 1995). This set of categories and properties was defined in such a way as to be able to provide the answers needed to navigate through the system network. Objects in application knowledge bases built in terms of this upper structure (by specializing its categories) are assured an interpretation in terms of the predicates that the systemic grammar needs because these are provided implicitly through the location of the objects in the taxonomy.

Mechanically, the process of generating a text using a systemic grammar consists of walking through the set of systems from the initial choice (which for a speech act might be whether it constitutes a statement, a question, or a command) through to its leaves, following several simultaneous paths through the system network until it has been completely traversed. Several parallel paths because in the analyses adopted by systemicists, the final shape of a text is dictated by three independent kinds of information: *experiential*, focusing on content; *interpersonal*, focusing on the interaction and stance toward the audience; and *textual*, focusing on form and stylistics.

As the network is traversed, a set of features that describe the text are accumulated. These may be used to ‘preselect’ some of the options at a lower ‘strata’ in the accumulating text, as for example when the structure of an embedded clause is determined by the traversal of the network that determines the functional organization of its parent clause. The features describing the subordinate’s function are passed through to what will likely be a recursive instantiation of the network that was traversed to form the parent, and they serve to fix the selection in key systems, for example, dictating that the clause should appear without an actor, for example, as a prepositionally marked gerund: “*You blocked me by taking the corner opposite mine.*”

The actual text takes shape by projecting the lexical realizations of the elements of the input specification onto selected positions in a large grid of possible positions as dictated by the features selected from the network. The words may be given by the final stages of the system network (as systemicists say: ‘lexis as most delicate grammar’) or as part of the input specification.

6.5.6 Functional Unification Grammars

Having a functional or purpose-oriented perspective in a grammar is largely a matter of the grammar’s content, not its architecture. What sets functional approaches to realization apart from structural approaches is the choice of terminology and distinctions, the indirect relationship to syntactic surface structure, and, when embedded in a realization component, the nature of its interface to the earlier text-planning components. Functional realizers are concerned with purposes, not contents. Just as a functional perspective can be implemented in a system network, it can be implemented in an annotated TAG (Yang et al. 1991) or, in what we will turn to now, in a unification grammar.

A unification grammar is also traversed, but this is less obvious since the traversal is done by the built-in unification process and is not something that its developers actively consider. (Except for reasons of efficiency, the early systems were notoriously slow because nondeterminism led to a vast amount of backtracking; as machines have gotten faster and the algorithms have been improved, this is no longer a

problem.)

The term ‘unification grammar’ emphasizes the realization mechanism used in this technique, namely merging the component’s input with the grammar to produce a fully specified, functionally annotated surface structure from which the words of the text are then read out. The merging is done using a particular form of unification; a thorough introduction can be found in McKeown (1985). In order to be merged with the grammar, the input must be represented in the same terms; it is often referred to as a ‘deep’ syntactic structure.

Unification is not the primary design element in these systems however, it just happened to be the control paradigm that was in vogue when the innovative data structure of these grammars — feature structures — was introduced by linguists as a reaction against the pure phrase structure approaches of the time (the late 1970s). Feature structures (FS) are much looser formalisms than unadorned phrase structures; they consist of sets of multilevel attribute-value pairs. A typical FS will incorporate information from (at least) three levels simultaneously: meaning, (surface) form, and lexical identities. FS allow general principles of linguistic structure to be stated more freely and with greater attention to the interaction between these levels than had been possible before.

The adaption of feature-structure-based grammars to generation was begun by Martin Kay (1984), who developed the idea of focusing on functional relationships in these systems — functional in the same sense as it is employed in systemic grammar, with the same attendant appeal to people working in generation who wanted to experiment with the feature-structure notation.

Kay’s notion of a ‘functional’ unification grammar (FUG) was first deployed by Appelt (1985), and then adopted by McKeown. McKeown’s students, particularly Michael Elhadad, made the greatest strides in making the formalism efficient. He developed the FUF system, which is now widely used (Elhadad 1991, Elhadad and Robin 1996). Elhadad also took the step of explicitly adopting the grammatical analysis and point of view of systemic grammarians, demonstrating quite effectively that grammars and the representations that embody them are separate aspects of system design.

6.6 The Cutting Edge

There has been a great deal of technical development in the last decade. For example, we have new surface realizers such as Mathew Stone’s SPUD (Stone et al. 2001) that works at the semantic and syntactic level simultaneously, or Michael White’s work based on the CCG grammar formalism (White and Baldridge 2003). Template-based realizers have also made a comeback (e.g., McRoy et al. 2003). And perhaps most of all there has been an massive influx of machine-learning-based machinery into the generation as there has in the rest of computational linguistics (see, e.g., Langkilde and Knight 1998, Bangalore and Rambow 2000).

However, for the most part these developments are just giving us better (or just alternative) ways of doing the things we already know how to do. In this final section I want to instead briefly describe two systems that are breaking entirely new ground.

6.6.1 Story Generation

The subject matter, or genre, of nearly all work is expository, providing explanations or simply conveying information. But much if not most of human talk is based on telling stories. Around the turn of the millennium Charles Callaway developed the StoryBook system (2002), which deployed the organizing principles of a rich model of narrative and the full panoply of generation facilities* to generate variations on the story of Little Red Riding Hood. Here is an excerpt.

* A narrative organizer that segmented and structured a the element of provided by the narrative planner, did lexical choice and maintained a discourse history; a sentence planner; a revision component; and a surface realizer (based on FUF) that knows how to format prose with embedded dialog.

Once upon a time a woodman and his wife lived in a pretty cottage on the borders of a great forest. They had one little daughter, a sweet child, who was a favorite with everyone. She was the joy of her mother's heart. To please her, the good woman made her a little scarlet cloak and hood. She looked so pretty in it that everyone called her Little Red Riding Hood.

StoryBook begins its substantive work at the start of the microplanning phase of generation after the content of could be said has been established and organized into a narrative stream by a simple FSA acting in lieu of a real narrative planner. The excerpt of this stream below draws on an ontology of concepts and relations that provides the raw material for the microlevel narrative planner.

```
(... ;; "once upon a time there was a woodman and his
(actor-property exist-being woodman001)
(refinement and-along-with woodman001 wife001)
(refinement belonging-to woodman001 wife001)
(specification exist-being process-step-type once-upon-a-time) )
```

Notice how lexical and 'close to the surface' the terms in this micro-planner input are. This permits the revision facilities in StoryBook to know enough about the individual abstract elements of the text to readily formulate highly composed prose.

6.6.2 Personality-Sensitive Generation

Remarkably few generation systems have been developed where the speaker could be said to have a particular personality. Clippinger and Brown's Erma certainly did, albeit at the cost of an intense, one-off programming effort. Eduard Hovy's Pauline (1990) was the first to show how this could be done systematically albeit just for exposition. First of all there must be a large number of relevant 'units' of content that could be included or ignored or systematically left to inference according to the desired level detail or choice of perspective. Second and more important is the use of a multilevel 'standoff' architecture whereby pragmatic notions ('use high style,' 'be brief') are progressively reinterpreted through one or more level of description as features that a generator can actually attend to (e.g., word choice, sentence length, clause complexity).

The currently most thorough and impressive treatment of personality in generation is François Mairesse and Marilyn Walker's Personage system (2007, 2008). Here are two generated examples in the domain of restaurant recommendation, one with a low extroversion rating and then one with a high rating.*

5 (2.83) Right, I mean, Le Marais is the only restaurant that is any good.

3 (6.0) I am sure you would like Le Marais, you know. The atmosphere is acceptable, the servers are nice and it's a french, kosher and steak house place. Actually, the food is good, even if its price is 44 dollars.

Personage is based on modeling the correlation of a substantial number of language variables (e.g., verbosity, repetition, filled pauses, stuttering) with personality as characterized by the Big Five personality traits. This model then drives a statistical microplanner (Stent et al. 2004) whose output is passed through a surface realizer based on Mel'cuk's *Meaning Text Theory of Language* (Lavoie and Rambow 1998). (Yet another example is the frequently eclectic combinations of theories and techniques that characterize work in computational linguistics because of the people doing the work and the accidents of history of who they studied with.)

* Utterance numbers and 1–7 extraversion ranking from Mairesse and Walker (2007, p. 496).

6.7 Conclusions

This chapter has covered the basic issues and perspectives that have governed work on natural language generation. With the benefit of hindsight it has tried to identify the axes that distinguish the different tacks people have taken during the last 40 years: does the speaker intentionally ‘push’ directives to the text planner, or does the planner ‘pull’ data out of a passive data base; does surface realization consist of ‘assembling’ a set of components or of ‘navigating’ through one large structure?

Given this past, what can we say about the future? One thing we can be reasonably sure of is that there will be relatively little work done on surface realization. People working on speech or doing computational psycholinguistics may see the need for new architectures at this level, and the advent of a new style of linguistic theory may prompt someone to apply it to generation; but most groups will elect to see realization as a solved problem — a complete module that they can ftp from a collaborating site.

By that same token, the linguistic sophistication and ready availability of the mature realizers (Penman, FUF) will mean that the field will no longer sustain abstract work in text planning; all planners will have to actually produce text, preferably pages of it, and that text should be of high quality. Toy output that neglects to properly use pronouns or is redundant and awkward will no longer be acceptable.

The most important scientific achievement to look toward in the course of the next 10 years is the emergence of a coherent consensus architecture for the presently muddled ‘middle ground’ of microplanning. Sitting between the point at which generation begins, where we have a strong working knowledge of how to fashion and deploy schemas, plan operators, and the like to select what is to be said and give it a coarse organization, and the point at which generation ends, where we have sophisticated, off-the-shelf surface realization components that one can use with only minimal personal knowledge of linguistics, we presently have a grab bag of phenomena that no two projects deal with in the same way (if they handle them at all).

In this middle ground lies the problem of where to use pronouns and other such reduced types of ‘subsequent reference’; the problem of how to select the best words to use (‘lexical choice’) and to pick among alternative paraphrases; and the problem how to collapse the set of propositions that the planner selects, each of which might be its own sentence if generated individually, into fluent complex sentences that are free of redundancy and fit the system’s stylistic goals (aggregation).

At this point, about all that is held in common in the community are the names of these problems and what their effects are in the final texts. That at least provides a common ground for comparing the proposals and systems that have emerged, but the actual alternatives in the literature tend to be so far apart in the particulars of their treatments that there are few possibilities for one group to build on the results of another.

To take just one example, it is entirely possible that aggregation — the present term of art in generation for how to achieve what others what others call ‘cohesion’ (Halliday and Hasan 1976) or just ‘fluency’ — is not a coherent notion. Consider that for aggregation to occur there must be separate, independent things to be aggregated. It might turn out that this is an artifact of the architecture of today’s popular text planners and not at all a natural kind, that is, something that is handled with the same procedures and at the same points in the processing for all the different instances of it that we see in real texts.

Whatever the outcome of such questions, we can be sure that they will be pursued vigorously by an ever-burgeoning number of people. The special interest group on generation (SIGGEN) has over 400 members, the largest of all the special interest groups under the umbrella of the Association for Computational Linguistics (ACL). The field is international in scope, with major research sites from Australia to Israel.

There is much challenging work remaining to be done that will keep those of us who work in this field engaged for years, more likely decades to come. Whether the breakthroughs will come from traditional grant funded research or from the studios and basements of game makers and entrepreneurs is impossible

to say. Whatever the future, this is the best part of the natural language problem in which to work.

References

- Andre, E., G. Herzog, and T. Rist (1988) On the simultaneous interpretation of real world image sequences and the natural language description: The system SOCCER, *Proceedings of the Eighth ECAI*, Munich, Germany, pp. 449–454.
- Appelt, D. (1985) *Planning English Sentences*, Cambridge University Press, Cambridge, U.K.
- Bangalore, S. and O. Rambow (2000) Exploiting a probabilistic hierarchical model for generation, *Proceedings of the Eighteenth Conference on Computational Linguistics (COLING)*, Saarbrücken, Germany, pp. 42–48.
- Barwise, J. and J. Perry (1983) *Situations and Attitudes*, MIT Press, Cambridge, MA.
- Bateman, J.A. (1997) Enabling technology for multilingual natural language: The KPML development environment, *Journal of Natural Language Engineering*, 3(1):15–55.
- Bateman, J.A., R. Henschel, and F. Rinaldi (1995) Generalized upper model 2.1: Documentation. Technical Report, GMD/Institute für Integrierte Publikations- und Informationssysteme, Darmstadt, Germany.
- Becker, J. (1975) The phrasal lexicon, *Proceedings of the TINLAP-I*, Cambridge, MA, ACM, pp. 60–64; also available as BBN Report 3081.
- Bourbeau, L., D. Carcagno, E. Goldberg, R. Kittredge, and A. Polguère (1990) Bilingual generation of weather forecasts in an operations environment, *COLING*, Helsinki, Finland.
- Brown, R. (1974) Use of multiple-body interrupts in discourse generation, Bachelor's thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA.
- Cahill, L., C. Doran, R. Evans, C. Mellish, D. Paiva, M.R.D. Scott, and N. Tipper (1999) In search of a reference architecture for NLP systems, *Proceedings of the European Workshop on Natural Language Generation*, Toulouse, France.
- Callaway, C., E. Not, A. Novello, C. Rocchi, O. Sock, and M. Zancanaro (2005) Automatic cinematography and multilingual NLG for generating video documentaries, *Artificial Intelligence*, 16(5): 57–89.
- Cassell, J., T. Bickmore, L. Campbell, H. Vilhjalmsson, and H. Yan (2000) Human conversation as a system framework: Designing embodied conversational agents, In J. Cassel, J. Sullivan, S. Prevost, and E. Churchill (eds.), *Embodied Conversational Agents*, MIT Press, Cambridge, MA.
- Clippinger, J. (1977) *Meaning and Discourse: A Computer Model of Psychoanalytic Speech and Cognition*, Johns Hopkins University Press, Baltimore, MA.
- Conklin E.J. and D. McDonald (1982) Saliency: The key to selection in deep generation, *Proceedings of the ACL-82*, University of Toronto, Toronto, ON, pp. 129–135.
- Cosby, N. (1999) SIMNET—An insider's perspective, *Simulation Technology* 2(1), http://www.sisostds.org/webletter/siso/iss_39/arg_202.htm, sampled 11/08.
- Dale, R., C. Mellish, and M. Zock (1990) *Current Research in Natural Language Generation*, Academic Press, Boston, MA.
- Dale, R., J. Oberlander, and M. Milosavljevic (1998) Integrating natural language and hypertext to produce dynamic documents, *Interacting with Computers*, 11(2):109–135.
- Davey, A. (1978) *Discourse Production*, Edinburgh University Press, Edinburgh, U.K.
- De Smedt, K. (1990) Incremental sentence generation, Technical Report 90-01, Nijmegen Institute for Cognition Research and Information Technology, Nijmegen, the Netherlands.
- Devlin, K. (1991) *Logic and Information*, Cambridge University Press, Cambridge, U.K.

- DiMarco, C. and G. Hirst (1993) A computational theory of goal-directed style in syntax, *Computational Linguistics*, 19(3):451–499.
- Elhadad, M. (1991) FUF: The universal unifier user manual (v5), Technical Report CUCS-038–91, Department of Computer Science, Columbia University, New York.

- Elhadad, M. and J. Robin (1996) An overview of SURGE: A reusable comprehensive syntactic realization component, Technical Report 96-03, Department of Mathematics and Computer Science, Ben Gurion University, Beer Sheva, Israel.
- Fawcett, R. (1981) Generating a sentence in systemic functional grammar, In M.A.K. Halliday and J.R. Martin (eds.), *Readings in Systemic Linguistics*, Batsford, London, U.K.
- Feiner, S. and K. McKeown (1991) Automating the generation of coordinated multimedia explanations, *IEEE Computer*, 24(10):33–40.
- Gabriel, R.P. (1981) An organization of programs in fluid domains, PhD thesis, Stanford; available as Stanford Artificial Intelligence Memo 342 (STAN-CA-81-856, 1981).
- Gabriel, R.P. (1986) Deliberate writing, In D.D. McDonald and L. Bolc (eds.), *Natural Language Generation Systems*, Springer-Verlag, New York, pp. 1–46.
- Geldof, S. (1996) Hyper-text generation from databases on the Internet, *Proceedings of the Second International Workshop on Applications of Natural Language to Information Systems, NLDB*, Amsterdam, the Netherlands, pp. 102–114, IOS Press.
- Green, S.J. and C. DiMarco (1996) Stylistic decision-making in natural language generation, In A. Givanni and M. Zock (eds.), *Trends in Natural Language Generation: An Artificial Intelligence Perspective, Lecture Notes in Artificial Intelligence*, 1036, Springer-Verlag, Berlin, Germany, pp. 125–143.
- Halliday, M.A.K. (1967) Notes on transitivity and theme in English Parts 1, 2, & 3, *Journal of Linguistics* 3.1, 3.2, 3.3: 37–81, 199–244, 179–215.
- Halliday, M.A.K. (1985) *An Introduction to Functional Grammar*, Edward Arnold, London, U.K.
- Halliday, M.A.K. and R. Hasan (1976) *Cohesion in English*, Longman, London, U.K.
- Halliday, M.A.K. and C.M.I.M. Matthiessen (2004) *An Introduction to Functional Grammar*, Edward Arnold, London, U.K.
- Hovy, E. (1990) Pragmatics and natural language generation, *Artificial Intelligence* 43:153–197.
- Kay, M. (1984) Functional unification grammar: A formalism for machine translation, *Proceedings of COLING-84*, Stanford, CA, pp. 75–78 ACL.
- Kittredge, R. and J. Lehrberger (1982) *Sublanguage: Studies of Language in Restricted Semantic Domains*, de Gruyter, Berlin, Germany.
- Kukich, K. (1988) Fluency in natural language reports, In D.D. McDonald and L. Bolc (eds.), *Natural Language Generation Systems*, Springer-Verlag, New York, pp. 280–312.
- Langkilde, I. and K. Knight (1998) Generation that exploits corpus-based statistical knowledge, *Proceedings of the ACL*, Montreal, Canada, pp. 704–710.
- Lavoie, B. and O. Rambow (1998) A fast and portable realizer for text generation systems, *Proceedings of the ANLP*, Washington, DC, ACL.
- Levelt, W.J.M. (1989) *Speaking*, MIT Press, Cambridge, MA.
- Loyall, A.B., W.S.N. Reilly, J. Bates and P. Weyhrauch (2004) System for authoring highly interactive, personality-rich interactive characters, *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, Grenoble, France.
- Mairesse, F. and M. Walker (2007) PERSONAGE: Personality generation for dialogue, *Proceedings of the Forty Fifth Annual Meeting of the Association for Computational Linguistics, ACL*, pp. 496–503.

- Mairesse, F. and M. Walker (2008) Trainable generation of big-five personality styles through data- driven parameter estimation, *Proceedings of the Forty Sixth Annual Meeting of the Association for Computational Linguistics, ACL*, pp.165–173.
- Mateas, M. and A. Stern (2002) A behavior language for story-based believable agents, *IEEE Intelligent Systems*, 17(4):39–47.
- Mateas, M. and A. Stern (2003) Façade: An experiment in building a fully-realized interactive drama, *Game Developers Conference, Game Design Track*, San Francisco, CA.
- Matthiessen, C.M.I.M. (1983) Systemic grammar in computation: The Nigel case, *Proceedings of the First Annual Conference of the European Chapter of the Association for Computational Linguistics*, Pisa, Italy.

- McDonald, D. and M. Meteer (1988) From water to wine: Generating natural language text from today's application programs, *Proceedings of the Second Conference on Applied Natural Language Processing (ACL)*, Austin, TX, pp.41–48.
- McDonald, D., M. Meteer, and J. Pustejovsky (1987) Factors contributing to efficiency in natural language generation, In G. Kempen (ed), *Natural Language Generation*, Martinus Nijhoff Publishers, Dordrecht, pp. 159–182.
- McKeown, K.R. (1985) *Text Generation*, Cambridge University Press, Cambridge, U.K.
- McKeown, K.R., M. Elhadad, Y. Fukumoto, J. Lim, C. Lombardi, J. Robin, and F. Smadja (1990) Natural language generation in COMET, In Dale et al., pp. 103–140.
- McRoy, S., S. Channarukul, and S. Ali (2003) An augmented template-based approach to text realization, *Natural Language Engineering*, 9(4):381–420.
- Meteer, M. (1992) *Expressibility and the Problem of Efficient Text Planning*, Pinter, London, U.K.
- Meteer, M., D. McDonald, S. Anderson, D. Forster, L. Gay, A. Huettner, and P. Sibun (1987) Mumble-86: Design and Implementation, Technical Report 87–87, Department of Computer and Information Science, University of Massachusetts at Amherst, Amherst, MA.
- Moore, J.D. and W.R. Swartout (1991) A reactive approach to explanation: Taking the user's feedback into account, In Paris et al., pp. 3–48.
- O'Donnell, M., C. Mellish, J. Oberlander, and A. Knott (2001) ILEX: An architecture for a dynamic hypertext generation system, *Natural Language Engineering*, 7(3):225–250.
- Paris, C.L. (1991) Generation and explanation: Building an explanation facility for the explainable expert systems framework, In C.L. Paris, W.R. Swartout, and W.C. Mann (eds.), *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, Kluwer Academic, Boston, MA, pp. 49–82.
- Penman Natural Language Group (1989) The Penman Documentation, USC Information Sciences Institute, Los Angeles, CA.
- Piwek, P., J. Masthoff, and M. Bergenstråle (2005) Reference and gestures in dialog generation: Three studies with embodied conversational agents, *AISP'05: Proceedings of the Joint Symposium on Virtual Social Agents*, Hatfield, U.K., pp. 53–60.
- Reiter, E., R. Robertson, and L.M. Osman (2003) Lessons from a failure: Generated tailored smoking cessation letters, *Artificial Intelligence*, 144:41–58.
- Reiter, E., A. Gatt, J. Hunter, S. Sripada, J. Hunter, J. Yu, and I. Davy (2005) Choosing words in computer generated weather forecasts, *Artificial Intelligence*, 167(1–2):137–169.
- Robin, J. (1993) A revision-based generation architecture for reporting facts in their historical context, In H. Horacek and M. Zock (eds.), *New Concepts in Natural Language Generation: Planning, Realization, and Systems*, Pinter, London, U.K., pp. 238–268.
- Robin, J. (1996) Evaluating the portability of revision rules for incremental summary generation, *Proceedings of the 34th Annual Meeting of the ACL*, Santa Cruz, CA, pp. 205–214.
- Sacerdoti, E. (1977) *A Structure for Plans and Behavior*, North-Holland, Amsterdam, the Netherlands.
- Stent, A., R. Prasad, and M. Walker (2004) Trainable sentence planning for complex information presentation in spoken dialog systems, *Proceedings of the 42nd Annual Meeting of the ACL*, Barcelona, Spain.
- Stock, O. and M. Zancanaro (2007) *PEACH: Intelligent Interfaces for Museum Visits*, Springer-Verlag, Berlin, Germany.
- Stone, M., C. Doran, B. Webber, T. Bleam, and M. Palmer (2001) Microplanning with communicative intentions: The SPUD system. Rutgers TR 65, distributed on arxiv.org.

- Streit, M., A. Batliner and T. Portele (2006) Emotional analysis and emotional-handling subdialogs, In W. Wahlster (ed), *Smartkom: Foundations of Multimodal Dialogue Systems*, Springer, Berlin, Germany.
- Swartout, W., J. Gratch, R. Hill, E. Hovy, S. Marsella, J. Rickel, and D. Traum (2006) Toward virtual humans, *AI Magazine*, 27(2):96–108.

- Traum, D., A. Roque, A. Leuski, P. Georgiou, J. Gerten, B. Martinovski, S. Narayanan, S. Robinson, and A. Vaswani (2007) Hassan: A virtual human for tactical questioning, *Proceedings of the Eighth SIGdial Workshop on Discourse and Dialog*, Antwerp, Belgium, pp. 71–74.
- Wallace, R.S. www.alicebot.org, see also en.wikipedia.org/wiki/A.L.I.C.E.
- Weizenbaum, J. (1966) ELIZA — A computer program for the study of natural language communication between man and machine, *Communications of the ACM*, 9(1):36–45.
- Wilcock, G. (1998) Approaches to surface realization with HPSG, *Ninth International Workshop on Natural Language Generation*, Niagara-on-the-Lake, Canada, pp. 218–227.
- Wilensky, R. (1976) Using plans to understand natural language, *Proceedings of the Annual Meeting of the Association for Computing Machinery*, Houston, TX.
- Winograd, T. (1972). *Understanding Natural Language*, Academic Press, New York.
- White, M. and J. Baldridge. (2003) Adapting chart realization to CCG, *Proceedings of the Ninth European Workshop on Natural Language Generation*, Toulouse, France.
- Yang, G., K.F. McCoy, and K. Vijay-Shanker (1991) From functional specification to syntactic structure: Systemic grammar and tree-adjointing grammar, *Computational Intelligence*, 7(4):207–219.

3. Social Robotics

Social robot

From Wikipedia, the free encyclopedia

[Jump to navigation](#) [Jump to search](#)

A **social robot** is an [autonomous robot](#) that interacts and communicates with humans or other autonomous physical agents by following social behaviors and rules attached to its role. Like other robots, a social [robot](#) is

physically embodied (avatars or on-screen synthetic social characters are not embodied and thus distinct). Some synthetic social agents are designed with a screen to represent the head or 'face' to dynamically communicate with users. In these cases, the status as a social robot depends on the form of the 'body' of the social agent; if the body has and uses some physical motors and sensor abilities, then the system could be considered a robot.



Contents

- [1 Background](#)
- [2 Definition](#)
- [3 Societal Impacts](#)
- [4 See also](#)
- [5 Further references](#)
- [6 References](#)
- [7 External links](#)

Background^{[\[edit\]](#)}

While robots have often been described as possessing social qualities (see for example the tortoises developed by [William Grey Walter](#) in the 1950s), social robotics is a fairly recent branch of robotics. Since the early 1990s [artificial intelligence](#) and [robotics](#) researchers have developed robots which explicitly engage on a social level. Notable researchers include [Cynthia Breazeal](#), Tony Belpaeme, Aude Billard, Kerstin Dautenhahn, Yiannis Demiris, [Hiroshi Ishiguro](#), Maja Mataric, Javier Movellan, Brian Scassellati and [Dean Weber](#). Also related is the Kansai engineering movement in Japanese science and technology --- for social robotics, see especially work by Takayuki Kanda, Hideki Kozima, [Hiroshi Ishiguro](#), Micho Okada, Tomio Watanabe, and P. Ravindra S. De Silva.

Designing an [autonomous](#) social robot is particularly challenging, as the robot needs to correctly interpret people's action and respond

appropriately, which is currently not yet possible. Moreover, people interacting with a social robot may hold very high expectancies of its capabilities, based on [science fiction](#) representations of advanced social robots. As such, many social robots are partially or fully [remote controlled](#) to simulate advanced capabilities. This method of (often covertly) controlling a social robot is referred to as a [Mechanical Turk](#) or [Wizard of Oz](#), after the character in the [L. Frank Baum](#) book. Wizard of Oz studies are useful in social robotics research to evaluate how people respond to social robots.

Definition^{[\[edit\]](#)}

A robot is defined in the International Standard of Organization as a reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for performance of a variety of tasks. As a subset of robots, social robots perform any or all of these processes in the context of a social interaction. The nature of the social interactions is immaterial and may range from relatively simple supportive tasks, such as passing tools to a worker, to complex expressive communication and collaboration, such as assistive healthcare.

Social interactions are likely to be cooperative, but the definition is not limited to this situation. Moreover, uncooperative behavior can be considered social in certain situations. The robot could, for example, exhibit competitive behavior within the framework of a game. The robot could also interact with a minimum or no communication. It could, for example, hand tools to an astronaut working on a space station. However, it is likely that some communication will be necessary at some point.

Two suggested^{[\[by whom?\]](#)} ultimate requirements for social robots are the [Turing Test](#) to determine the robot's communication skills and [Isaac Asimov's Three Laws of Robotics](#) for its behavior (The usefulness to apply these requirements in a real-world application, especially in the case of Asimov's laws, still is disputed^{[\[by whom?\]](#)} and may not be possible at all). However, a

consequence of this viewpoint is that a robot that only interacts and communicates with other robots would not be considered to be a social robot: Being social is bound to humans and their society which defines necessary social values, norms and standards.^[1] This results in a cultural dependency of social robots since social values, norms and standards differ between cultures.

This brings us directly to the last part of the definition. A social robot must interact within the social rules attached to its role. The role and its rules are defined through society. For example, a robotic butler for humans would have to comply with established rules of good service. It should be anticipating, reliable and most of all discreet. A social robot must be aware of this and comply with it. However, social robots that interact with other autonomous robots would also behave and interact according to non-human conventions. In most social robots the complexity of human-to-human interaction will be gradually approached with the advancement of the technology of [androids](#) (a form of [humanoid robots](#)) and implementation of a variety of more human-like communication skills ^[2]

An example of social robot would be [Kaspar](#), designed by [University of Hertfordshire](#) to help children with autism learn responses from the robot through games and interactive play.^[3]

Societal Impacts^[edit]

The increasingly widespread use of more advanced social robots is one of several phenomena expected to contribute to the technological [posthumanization](#) of human societies, through which process “a society comes to include members other than ‘natural’ biological human beings who, in one way or another, contribute to the structures, dynamics, or meaning of the society.”^[4]

See also^[edit]

-  [Robotics portal](#)

- [hitchBOT](#) (decapitated in Philadelphia)
- [Kismet](#)
- [Joe Robot](#)
- [Tico](#)
- [ElliQ](#)

Further references[\[edit\]](#)



This article includes a [list of references](#), but **its sources remain unclear** because it has **insufficient inline citations**. Please help to [improve](#) this article by [introducing](#) more precise citations. *(June 2011)* ([Learn how and when to remove this template message](#))

- Walter, W. Grey (May 1950). "An Imitation of Life". *Scientific American*. pp. 42–45.
- Dautenhahn, Kerstin (1994). Gaussier, P.; Nicoud, J. D., eds. "Trying to Imitate - a Step Towards Releasing Robots from Social Isolation". *Proceedings: From Perception to Action Conference*. Lausanne, Switzerland: IEEE Computer Society Press: 290–301,. [ISBN 0-8186-6482-7](#).
- Dautenhahn, Kerstin (1995). "Getting to know each other - artificial social intelligence for autonomous robots". *Robotics and Autonomous Systems*. **16**: 333–356.
- Breazeal, Cynthia L. (2002). *Designing Sociable Robots*. MIT Press. [ISBN 0-262-02510-8](#).
- Fong, Terrence; Nourbakhsh, Illah R.; Dautenhahn, Kerstin (2003). "A survey of socially interactive robots". *Robotics and Autonomous Systems*. **42** (3-4): 143–166.
- Duffy, Brian R. (2004). "Social Embodiment in Autonomous Mobile Robotics". *International Journal of Advanced Robotic Systems*. **1** (3): 155–170.

- *Duffy, Brian R. (2008). "Fundamental Issues in Affective Intelligent Social Machines". Open Artificial Intelligence Journal. 2 (14): 21–34. ISSN 1874-0618.*

Nadine Social Robot

From Wikipedia, the free encyclopedia

[Jump to navigation](#) [Jump to search](#)



This article's **tone or style may not reflect the [encyclopedic tone](#) used on Wikipedia**. See Wikipedia's [guide to writing better articles](#) for suggestions. *(February 2018)* ([Learn how and when to remove this template message](#))



This article **relies too much on [references to primary sources](#)**. Please improve this by adding [secondary or tertiary sources](#). *(February 2018)* ([Learn how and when to remove this template message](#))

Nadine	
	
Year of creation	2013

Nadine is a female humanoid [social robot](#) that is modelled on [Professor Nadia Magnenat Thalmann](#). The robot has a strong human-likeness with a natural-looking skin and hair^{[[1](#)][[2](#)][[3](#)]} and realistic hands.^{[[4](#)][[5](#)][[6](#)][[7](#)][[8](#)][[9](#)][[10](#)][[11](#)]} Nadine is a socially intelligent robot which returns a greeting, makes eye contact, and

remembers all the conversations had with it. It is able to answer questions autonomously in several languages, simulate emotions both in gestures and facially, depending on the content of the interaction with the user.^{[12][13][14]} Nadine can recognise persons it has previously seen, and engage in flowing conversation.^{[15][16][17][18]} Nadine has been programmed with a "personality", in that its demeanour can change according to what is said to it.^[19] Nadine has a total of 27 [degrees of freedom](#) for facial expressions and upper body movements. With persons it has previously encountered, it remembers facts and events related to each person.^[20] It can assist people with special needs by reading stories, showing images, put on [Skype](#) sessions, send emails, and communicate with other members of the family.^{[21][22][23][24]} It can play the role of a receptionist in an office or be dedicated to be a personal coach.^{[25][26]} Nadine interacted with more than 100,000 visitors at the ArtScience Museum in Singapore during the exhibition, "HUMAN+: The Future of our Species", that was held from May to October 2017.^{[27][28][29]}

Contents

[hide]

- [1_Platform](#)
- [2_Specifications](#)
- [3_References](#)
- [4_External links](#)

Platform[\[edit\]](#)

Nadine’s platform is implemented as a classic Perception-Decision-Action architecture. The perception layer is composed of a [Microsoft Kinect](#) V2 and a microphone. The perception includes face recognition, gestures recognition^[30] and some understanding of social situations. In regard to decision, the platform includes emotion and memory models as well as social attention. Finally, the action layer consists of a dedicated robot

controller which includes emotional expressions, lips synchronization and online gaze generation.

Social robot

A **social robot** is an [autonomous robot](#) that interacts and communicates with humans or other autonomous physical agents by following social behaviors and rules attached to its role. Like other robots, a social [robot](#) is physically embodied (avatars or on-screen synthetic social characters are not embodied and thus distinct). Some synthetic social agents are designed with a screen to represent the head or 'face' to dynamically communicate with users. In these cases, the status as a social robot depends on the form of the 'body' of the social agent; if the body has and uses some physical motors and sensor abilities, then the system could be considered a robot.

Background

While robots have often been described as possessing social qualities (see for example the tortoises developed by [William Grey Walter](#) in the 1950s), social robotics is a fairly recent branch of robotics. Since the early 1990s [artificial intelligence](#) and [robotics](#) researchers have developed robots which explicitly engage on a social level. Notable researchers include [Cynthia Breazeal](#), Tony Belpaeme, Aude Billard, Kerstin Dautenhahn, Yiannis Demiris, Maja Mataric, Javier Movellan, Brian Scassellati and [Dean Weber](#). Also related is the Kansai engineering movement in Japanese science and technology --- for social robotics, see especially work by Takayuki Kanda, Hideki Kozima, [Hiroshi Ishiguro](#) and Tomio Watanabe.

Designing an [autonomous](#) social robot is particularly challenging, as the robot needs to correctly interpret people's action and respond appropriately, which is currently not yet possible.

Moreover, people interacting with a social robot may hold very

high expectancies of its capabilities, based on [science fiction](#) representations of advanced social robots. As such, many social robots are partially or fully [remote controlled](#) to simulate advanced capabilities. This method of (often covertly) controlling a social robot is referred to as [Wizard of Oz](#), after the character in the [L. Frank Baum](#) book. Wizard of Oz studies are useful in social robotics research to evaluate how people respond to social robots.

Definition

A robot is defined in the International Standard of Organization as a reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for performance of a variety of tasks. As a subset of robots, social robots perform any or all of these processes in the context of a social interaction. The nature of the social interactions is immaterial and may range from relatively simple supportive tasks, such as passing tools to a worker, to complex expressive communication and collaboration, such as assistive healthcare.

Social interactions are likely to be cooperative, but the definition is not limited to this situation. Moreover, uncooperative behavior can be considered social in certain situations. The robot could, for example, exhibit competitive behavior within the framework of a game. The robot could also interact with a minimum or no communication. It could, for example, hand tools to an astronaut working on a space station. However, it is likely that some communication will be necessary at some point.

Two suggested ultimate requirements for social robots are the [Turing Test](#) to determine the robot's communication skills and [Isaac Asimov's Three Laws of Robotics](#) for its behavior (The usefulness to apply these requirements in a real-world application, especially in the case of Asimov's laws, still is disputed and may not be possible at all). However, a consequence of this viewpoint is that a robot that only interacts and communicates with other robots would not be considered to

be a social robot: Being social is bound to humans and their society which defines necessary social values, norms and standards. This results in a cultural dependency of social robots since social values, norms and standards differ between cultures. This brings us directly to the last part of the definition. A social robot must interact within the social rules attached to its role. The role and its rules are defined through society. For example, a robotic butler for humans would have to comply with established rules of good service. It should be anticipating, reliable and most of all discreet. A social robot must be aware of this and comply with it. However, social robots that interact with other autonomous robots would also behave and interact according to non-human conventions. In most social robots the complexity of human-to-human interaction will be gradually approached with the advancement of the technology of [androids](#) (a form of [humanoid robots](#)) and implementation of a variety of more human-like communication skills ^[4]

An example of social robot would be Kaspar, designed by [University of Hertfordshire](#) to help children with autism learn responses from the robot through games and interactive play.^[16]

See also

- [Kismet](#)
- [Joe Robot](#)
- [Tico](#)

Sophia (robot)

From Wikipedia, the free encyclopedia

[Jump to navigation](#)

[Jump to search](#)

Sophia at ITU's AI for Good Global Summit in Geneva in May 2018

Sophia is a social humanoid robot developed by Hong Kong-based company Hanson Robotics. Sophia was activated on April 19, 2015[1] and made her first public appearance at South by Southwest Festival (SXSW) in mid-March 2016 in Austin, Texas, United States.[2] She is able to display more than 50 facial expressions.

Sophia has been covered by media around the globe and has participated in many high-profile interviews. In October 2017, Sophia, the robot became the first robot to receive citizenship of any country.[3][4] In November 2017, Sophia was named the United Nations Development Programme's first ever Innovation Champion, and the first non-human to be given any United Nations title.[5]

Contents

1

History

2

Features

3

Public figure

4

Events

5

Controversy over hype in the scientific community

6

See also

7

References

8

External links

History

Sophia was created by Hanson Robotics in collaboration with AI developers.

Sophia was activated on April 19, 2015.[6] The robot, modeled after actress Audrey Hepburn,[7] is known for her human-like appearance and behavior compared to previous robotic variants. According to the manufacturer, David Hanson, Sophia uses artificial intelligence, visual data processing and facial recognition. Sophia also imitates

human gestures and facial expressions and is able to answer certain questions and to make simple conversations on predefined topics (e.g. on the weather).[8] Sophia uses voice recognition (speech-to-text) technology from Alphabet Inc. (parent company of Google) and is designed to get smarter over time.[6] Sophia's intelligence software is designed by SingularityNET.[9] The AI program analyses conversations and extracts data that allows her to improve responses in the future.[10]

Hanson designed Sophia to be a suitable companion for the elderly at nursing homes, or to help crowds at large events or parks. He has said that he hopes that the robot can ultimately interact with other humans sufficiently to gain social skills.[3]

Sophia has seven robot humanoid “siblings” who were also created by Hanson Robotics.[11] Fellow Hanson robots are Alice, Albert Einstein Hubo, BINA48, Han, Jules, Professor Einstein, Philip K. Dick Android, Zeno,[11] and Joey Chaotic.[12]

Features

Sophia's internals

Cameras within Sophia's eyes combined with computer algorithms allow her to see. She can follow faces, sustain eye contact, and recognize individuals. She is able to process speech and have conversations using a natural language subsystem.[1] Around January 2018 Sophia was upgraded with functional legs and the ability to walk.[13]

Sophia is conceptually similar to the computer program ELIZA, which was one of the first attempts at simulating a human conversation.[14] The software has been programmed to give pre-written responses to specific questions or phrases, like a chatbot. These responses are used to create the illusion that the robot is able to understand conversation, including stock answers to questions like "Is the door open or shut?"[15] The information is shared in a cloud network which allows input and responses to be analysed with blockchain technology.[16]

David Hanson has said that Sophia would ultimately be a good fit to serve in healthcare, customer service, therapy and education.[17] Sophia runs on artificially intelligent software that is constantly being trained in the lab, so her conversations are likely to get faster, Sophia's expressions are likely to have fewer errors, and she should answer increasingly complex questions with more accuracy.[18]

Public figure

On November 21, 2017 Sophia was named the United Nations Development Programme's first ever Innovation Champion for Asia and the Pacific.[4] The announcement was made at the Responsible Business Forum in Singapore, an event hosted by the UNDP in Asia and the Pacific and Global Initiatives. As part of her role, Sophia will help to unlock innovation to work toward achieving the United Nation's Sustainable Development Goals. On stage, she was assigned her first task by UNDP Asia Pacific Chief of Policy and Program, Jaco Cilliers.[19]

Sophia has appeared on CBS 60 Minutes with Charlie Rose,[20] Good Morning Britain with Piers Morgan,[21] and outlets like CNBC, Forbes, Mashable, the New York Times, the Wall Street Journal, the Guardian, and the Tonight Show with Jimmy Fallon. Sophia was featured in AUDI's annual report[22] and was on the cover of ELLE Brasil magazine.[23] Sophia has also appeared in videos and music videos, including The White King, and as the lead female character in pop singer Leehom Wang's music video A.I.[24]

Events

Sophia speaking to a crowd, 2017

Sophia with Mukhisa Kituyi, Houlin Zhao and Tedros Adhanom Ghebreyesus in 2018

Sophia has been interviewed in the same manner as a human, striking up conversations with hosts. Some replies have been nonsensical, while others have impressed interviewers such as 60 Minutes' Charlie Rose.[10] In a piece for CNBC, when the interviewer expressed concerns about robot behavior, Sophia joked that he had "been reading too much Elon Musk. And watching too many Hollywood movies".[25] Musk tweeted that Sophia should watch The Godfather and asked "what's the worst that could happen?"[26][27] Business Insider's chief UK editor Jim Edwards interviewed Sophia, and while the answers were "not altogether terrible", he predicted it was a step towards "conversational artificial intelligence".[28] At the 2018 Consumer Electronics Show, a BBC News reporter described talking with Sophia as "a slightly awkward experience".[29]

On October 11, 2017, Sophia was introduced to the United Nations with a brief conversation with the United Nations Deputy Secretary-General, Amina J. Mohammed.[30] On October 25, at the Future Investment Summit in Riyadh, the robot was granted Saudi Arabian citizenship, becoming the first robot ever to have a nationality.[27][31] This attracted controversy as some commentators wondered if this implied that Sophia could vote or marry, or whether a deliberate system shutdown could be considered murder. Social media users used Sophia's citizenship to criticize Saudi Arabia's human rights record.[32][33] As expressed by Ali Al-Ahmed, director of the Institute for Gulf Affairs, "Women (in Saudi Arabia) have since committed suicide because they couldn't leave the house, and Sophia is running around [without a male guardian]. Saudi law doesn't allow non-Muslims to get citizenship. Did Sophia convert to Islam? What is the religion of this Sophia and why isn't she wearing hijab? If she applied for citizenship as a human she wouldn't get it."[34][35] It has been countered that not being a Muslim is not a formal barrier to being a Saudi citizen, though officials may take it into consideration at their discretion.[36] In December 2017, Sophia's creator David Hanson said in an interview that Sophia will use her citizenship to

advocate for women's rights in her new country of citizenship; Newsweek criticized that "What [Hanson] means, exactly, is unclear".[37]

Controversy over hype in the scientific community

According to Quartz, experts who have reviewed the robot's open-source code state that Sophia is best categorized as a chatbot with a face.[15] Many experts in the AI field disapprove of Sophia's overstated presentation. Ben Goertzel, the chief scientist for the company that made Sophia, acknowledges that it is "not ideal" that some think of Sophia as having human-equivalent intelligence, but argues Sophia's presentation conveys something unique to audiences: "If I show them a beautiful smiling robot face, then they get the feeling that 'AGI' (artificial general intelligence) may indeed be nearby and viable... None of this is what I would call AGI, but nor is it simple to get working." Goertzel added that Sophia does utilize AI methods including face tracking, emotion recognition, and robotic movements generated by deep neural networks. Sophia's dialogue is generated via a decision tree, but is integrated with these outputs uniquely.[38]

According to The Verge, Hanson often exaggerates and "grossly misleads" about Sophia's capacity for consciousness, for example by agreeing with Jimmy Fallon[15][39] in 2017 that Sophia was "basically alive".[38] In a piece produced by CNBC which indicates that their own interview questions for Sophia were heavily rewritten by her creators, Goertzel responds to the Hanson quote by suggesting Hanson means Sophia is "alive" in the way that, to a sculptor, a piece of sculpture becomes "alive" in the sculptor's eyes as the work nears completion.[40]

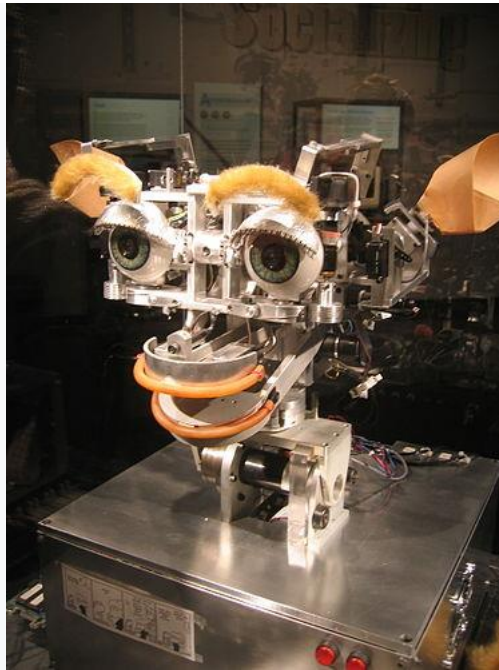
In January 2018, Facebook's director of artificial intelligence, Yann LeCun, tweeted that Sophia was "complete bullshit" and slammed the media for giving coverage to "Potemkin AI". In response, Goertzel stated that he had never pretended Sophia was close to human-level intelligence.[41]

Kismet (robot)

From Wikipedia, the free encyclopedia

[Jump to navigation](#) [Jump to search](#)

This article is about a robot called Kismet. For other uses of "kismet", see [Kismet](#).



Kismet now resides at the MIT Museum in [Cambridge, Massachusetts](#), United States.

Kismet is a [robot](#) head made in the late 1990s at [Massachusetts Institute of Technology](#) by Dr. [Cynthia Breazeal](#) as an experiment in [affective computing](#); a machine that can recognize and simulate [emotions](#). The name Kismet comes from a [Turkish](#) word meaning "fate" or sometimes "luck".^[1]

Hardware design and construction^[edit]

In order for Kismet to properly interact with human beings, it contains input devices that give it [auditory](#), [visual](#), and [proprioception](#) abilities. Kismet simulates emotion through various facial expressions, vocalizations, and movement. Facial expressions are created through movements of the ears, eyebrows, eyelids, lips, jaw, and head. The cost of physical materials is an estimated US\$25,000.^[1]

In addition to the equipment mentioned above, there are four [Motorola 68332s](#), nine 400 MHz PCs, and another 500 MHz PC.^[1]

Software system^[edit]

Kismet's social intelligence software system, or synthetic nervous system (SNS), was designed with human models of intelligent behavior in mind. It contains six subsystems^[2] as follows.

Low-level feature extraction system[\[edit\]](#)

This system processes raw visual and auditory information from cameras and microphones. Kismet's **vision system** can perform eye detection, motion detection and, albeit controversial, skin-color detection. Whenever Kismet moves its head, it momentarily disables its motion detection system to avoid detecting self-motion. It also uses its stereo cameras to estimate the distance of an object in its visual field, for example to detect threats—large, close objects with a lot of movement.^[3]

Kismet's **audio system** is mainly tuned towards identifying affect in [infant-directed speech](#). In particular, it can detect five different types of affective speech: approval, prohibition, attention, comfort, and neutral. The affective intent classifier was created as follows. Low-level features such as pitch mean and energy (volume) variance were extracted from samples of recorded speech. The classes of affective intent were then modeled as a [gaussian mixture model](#) and trained with these samples using the [expectation-maximization algorithm](#). Classification is done with multiple stages, first classifying an utterance into one of two general groups (e.g. soothing/neutral vs. prohibition/attention/approval) and then doing more detailed classification. This architecture significantly improved performance for hard-to-distinguish classes, like *approval* ("You're a clever robot") versus *attention* ("Hey Kismet, over here").^[3]

Motivation system[\[edit\]](#)

Dr. Breazeal figures her relations with the robot as 'something like an infant-caretaker interaction, where I'm the caretaker essentially, and the robot is like an infant'. The overview sets the human-robot relation within a frame of learning, with Dr. Breazeal providing the scaffolding for Kismet's development. It offers a demonstration of Kismet's capabilities, narrated as emotive facial expressions that communicate the robot's 'motivational state', Dr. Brazeal: "This one is anger (laugh) extreme anger, disgust,

excitement, fear, this is happiness, this one is interest, this one is sadness, surprise, this one is tired, and this one is sleep."^[4]

At any given moment, Kismet can only be in one emotional state at a time. However, Breazeal states that Kismet is not conscious, so it does not have feelings.^[5]

Motor system^[edit]

Kismet speaks a proto-language with a variety of phonemes, similar to baby's babbling. It uses the [DECTalk](#) voice synthesizer, and changes pitch, timing, articulation, etc. to express various emotions. Intonation is used to vary between question and statement-like utterances. Lip synchronization was important for realism, and the developers used a strategy from animation:^[6] "simplicity is the secret to successful lip animation". Thus, they did not try to imitate lip motions perfectly, but instead "create a visual short hand that passes unchallenged by the viewer".

See also^[edit]



- [*Robotics portal*](#)



- [*Artificial intelligence portal*](#)

- [Affective computing](#)
- [Artificial intelligence](#)

hitchBOT

From Wikipedia, the free encyclopedia

[Jump to navigation](#) [Jump to search](#)



hitchBOT displayed at an exhibition

hitchBOT was a Canadian "[hitchhiking robot](#)" created by David Harris Smith of [McMaster University](#) and Frauke Zeller of [Ryerson University](#).^[1] It gained international attention for successfully hitchhiking across Canada and in Europe, but in 2015 an attempt to hitchhike across the [United States](#) ended shortly after it began when the robot was destroyed by vandals in [Philadelphia](#), [Pennsylvania](#).^[2]

Contents

[hide]

- [1_Description](#)
- [2_Travels](#)
- [3_Destruction](#)
- [4_Legacy](#)
- [5_Other social experiment robots](#)
- [6_References](#)

Description[\[edit\]](#)



Original hitchBOT in collections.

The robot had a cylindrical body composed mainly from a plastic bucket, with two flexible "arms" and two flexible "legs" attached to the torso. The top section of the cylindrical body was transparent, containing a screen which displayed eyes and a mouth, making the robot approximately humanoid in external appearance.^[2]

The robot was able to carry on basic conversation and talk about facts, and was designed to be a robotic travelling companion while in the vehicle of the driver who picked it up. It had a [GPS](#) device and a [3G](#) connection, which allowed researchers to track its location. It was equipped with a camera, which took photographs periodically to document its journeys. It was powered either by solar power or by cigarette lighter sockets in cars.^[2]

The robot was not able to walk – it completed its "hitchhiking" journeys by "asking" to be carried by those who picked it up.

It was created as a social experiment. The robot's "hitchhiking" was reported by the press in many countries.^[2]

Travels^[edit]

From July 27, 2014 to August 21, 2014, it hitchhiked across Canada from the Institute for Applied Creativity at [NSCAD University](#)^[8] [Halifax, Nova Scotia](#), to [Victoria, British Columbia](#).^{[4][5]}

A second hitchBOT machine was made,^[6] and in February 2015 it was hitchhiking around [Germany](#) for a few days.^[7]

HitchBOT attempted to cross the United States from Boston to San Francisco starting in July 2015.

Destruction[\[edit\]](#)

On August 1, 2015 a photo^[8] was tweeted, showing that the robot had been decapitated in [Philadelphia, Pennsylvania](#).

On August 3, 2015 Adam Gabbatt reported on the Guardian website about hitchBOT's destruction. Frauke Zeller, co-creator of hitchBOT, was quoted saying:

"We can see on all our data that the tablet and battery and everything shut off at the same time so it must have been when they vandalised the bot,"

The article also states:

She said Hitchbot's body had been found by some good samaritans who had located the roving robot through a regularly updated map on its website. "They sent us images and it's really beyond repair. There's not a single wire inside and all the things are broken."^[9]

Legacy[\[edit\]](#)

The first hitchBOT is a permanent exhibit at the [Canada Science and Technology Museum](#).^[6]

Other social experiment robots[\[edit\]](#)

Zeller has also created an art critic robot called kulturBOT.^[10]

Media Lab Europe's social robots

From Wikipedia, the free encyclopedia

(Redirected from [Joe Robot](#))

[Jump to navigation](#) [Jump to search](#)

Joe Robot and **Anthropos** were [social robots](#) at [Media Lab Europe](#). They were developed to aid in research on [anthropomorphism](#) and to explore the illusion of life and intelligence during the development of a meaningful social interaction between artificial systems and people. As artificial systems enter our social space, we will inherently project/impose our interpretation on their actions similar to the techniques we employ in rationalising, for example, a pet's behaviour. This propensity to anthropomorphise should not be seen as a hindrance to social system development, but rather a useful mechanism that requires judicious examination and employment in social system research. A fundamental balance between function and form for human-robot interaction is necessary.

Tico Robot

From Wikipedia, the free encyclopedia

[Jump to navigation](#) [Jump to search](#)

Tico	
Manufacturer	Adele Robots
Year of creation	2007
Website	www.adelerobots.com/en/tico

Tico is a [social robot](#) developed by [Adele Robots](#) to interact with [humans](#) in different environments, primarily promotional events and in education as a helper for [teachers](#).

Contents

[hide]

- 1 [History](#)
- 2 [Design](#)
- 3 [Tico usage](#)

- 4 [See also](#)
- 5 [References](#)
- 6 [External links](#)

History^[edit]

Initially created by Treelogic in 2007 as a research platform for Human-Robot Interaction, it participated in several [R&D](#) projects such as [ACROSS](#),^[1] and was also tested in supermarkets and schools. It became commercially available in 2010.

Design^[edit]

The first version included a [laser rangefinder](#) and [ultrasonic sensors](#) for obstacle detection, and a camera for tracking people and reading [QR codes](#).^[clarification needed] It also has a touch screen. Its head and neck access 6 [degrees of freedom](#). It moves using [differential steering](#) with two high-torque [EC gear-motors](#), and runs [Ubuntu](#) 9.10 on a 1.6Ghz [Intel Core Duo](#). It stands 1.5 metres (4 ft 11 in) tall and weighs 85 kilograms (187 lb). A new, fresh battery allows it to run freely for eight hours.

Tico usage^[edit]

Tico was tested as a [pedagogic](#) tool for teachers to improve children's motivation and gain their attention.^[2]

See also^[edit]

- [Kismet](#)
- [Joe Robot](#)

4. Research Overview

Multimodal interaction

Multimodal interaction provides the user with multiple modes of interacting with a system. A **multimodal** interface provides several distinct tools for [input and output](#) of data. For example, a multimodal [question answering system](#) employs multiple modalities (such as text and photo) at both question (input) and answer (output) level.^[1]



Contents

- [1 Introduction](#)
- [2 Multimodal input](#)
- [3 Multimodal input and output](#)
- [4 Multimodal fusion](#)
- [5 Ambiguity](#)
- [6 See also](#)
- [7 References](#)
- [8 External links](#)

Introduction[[edit](#)]

Multimodal human-computer interaction refers to the "interaction with the virtual and physical environment through natural modes of communication",^[2] This implies that multimodal interaction enables a more free and natural communication, interfacing users with automated systems in both input and output.^[3] Specifically, multimodal systems can offer a flexible, efficient and usable environment allowing users to interact through input modalities, such as speech, handwriting, hand gesture and gaze, and to receive information by the system through output modalities, such as speech synthesis, smart graphics and other modalities, opportunely combined. Then a multimodal system has to

recognize the inputs from the different modalities combining them according to temporal and contextual constraints^[4] in order to allow their interpretation. This process is known as multimodal fusion, and it is the object of several research works from nineties to now.^{[5][6][7][8][9][10][11][12]} The fused inputs are interpreted by the system. Naturalness and flexibility can produce more than one interpretation for each different modality (channel) and for their simultaneous use, and they consequently can produce multimodal ambiguity^[13] generally due to imprecision, noises or other similar factors. For solving ambiguities, several methods have been proposed.^{[14][15][16][17][18][19]} Finally the system returns to the user outputs through the various modal channels (disaggregated) arranged according to a consistent feedback (fission).^[20] The pervasive use of mobile devices, sensors and web technologies can offer adequate computational resources to manage the complexity implied by the multimodal interaction. "Using cloud for involving shared computational resources in managing the complexity of multimodal interaction represents an opportunity. In fact, cloud computing allows delivering shared scalable, configurable computing resources that can be dynamically and automatically provisioned and released".^[21]

Multimodal input[\[edit\]](#)

Two major groups of multimodal interfaces have merged, one concerned in alternate input methods and the other in combined input/output. The first group of interfaces combined various user input modes beyond the traditional [keyboard](#) and [mouse input/output](#), such as speech, pen, touch, manual gestures,^[22] gaze and head and body movements.^[23] The most common such interface combines a visual modality (e.g. a display, keyboard, and mouse) with a voice [modality](#) ([speech recognition](#) for input, [speech synthesis](#) and recorded audio for output). However other modalities, such as pen-based input or [haptic](#) input/output may be used. Multimodal user interfaces are a research area in [human-computer interaction](#) (HCI).

The advantage of multiple input [modalities](#) is increased [usability](#).^[24] the weaknesses of one modality are offset by the strengths of another. On a mobile device with a small visual interface and keypad, a word may be quite difficult to type but very easy to say (e.g. [Poughkeepsie](#)). Consider

how you would access and search through [digital media](#) catalogs from these same devices or set top boxes. And in one real-world example, patient information in an operating room environment is accessed verbally by members of the surgical team to maintain an antiseptic environment, and presented in near realtime aurally and visually to maximize comprehension.

Multimodal input user interfaces have implications for [accessibility](#).^[25] A well-designed multimodal application can be used by people with a wide variety of impairments. Visually impaired users rely on the voice modality with some keypad input. Hearing-impaired users rely on the visual modality with some speech input. Other users will be "situationally impaired" (e.g. wearing gloves in a very noisy environment, driving, or needing to enter a credit card number in a public place) and will simply use the appropriate modalities as desired. On the other hand, a multimodal application that requires users to be able to operate all modalities is very poorly designed.

The most common form of input multimodality in the market makes use of the [XHTML+Voice](#) (aka X+V) Web markup language, an open [specification](#) developed by [IBM](#), [Motorola](#), and [Opera Software](#). [X+V](#) is currently under consideration by the [W3C](#) and combines several [W3C Recommendations](#) including [XHTML](#) for visual markup, [VoiceXML](#) for voice markup, and [XML Events](#), a standard for integrating [XML](#) languages. [Multimodal browsers](#) supporting [X+V](#) include IBM WebSphere Everyplace Multimodal Environment, [Opera](#) for [Embedded Linux](#) and [Windows](#), and [ACCESS Systems NetFront](#) for [Windows Mobile](#). To develop multimodal applications, [software developers](#) may use a [software development kit](#), such as IBM WebSphere Multimodal Toolkit, based on the [open source Eclipse framework](#), which includes an [X+V debugger](#), [editor](#), and [simulator](#).

Multimodal input and output^{[[edit](#)]}

The second group of multimodal systems presents users with multimedia displays and multimodal output, primarily in the form of visual and auditory cues. Interface designers have also started to make use of other modalities, such as touch and olfaction. Proposed benefits of multimodal output system include synergy and redundancy. The

information that is presented via several modalities is merged and refers to various aspects of the same process. The use of several modalities for processing exactly the same information provides an increased bandwidth of information transfer .^{[26][27][28]} Currently, multimodal output is used mainly for improving the mapping between communication medium and content and to support attention management in data-rich environment where operators face considerable visual attention demands.^[29]

An important step in multimodal interface design is the creation of natural mappings between modalities and the information and tasks. The auditory channel differs from vision in several aspects. It is omnidirectional, transient and is always reserved.^[29] Speech output, one form of auditory information, received considerable attention. Several guidelines have been developed for the use of speech. Michaelis and Wiggins (1982) suggested that speech output should be used for simple short messages that will not be referred to later. It was also recommended that speech should be generated in time and require an immediate response.

The sense of touch was first utilized as a medium for communication in the late 1950s.^[30] It is not only a promising but also a unique communication channel. In contrast to vision and hearing, the two traditional senses employed in HCI, the sense of touch is proximal: it senses objects that are in contact with the body, and it is bidirectional in that it supports both perception and acting on the environment.

Examples of auditory feedback include auditory icons in computer operating systems indicating users' actions (e.g. deleting a file, open a folder, error), speech output for presenting navigational guidance in vehicles, and speech output for warning pilots on modern airplane cockpits. Examples of tactile signals include vibrations of the turn-signal lever to warn drivers of a car in their blind spot, the vibration of auto seat as a warning to drivers, and the [stick shaker](#) on modern aircraft alerting pilots to an impending stall.^[29]

Invisible interface spaces became available using sensor technology. Infrared, ultrasound and cameras are all now commonly used.^[31] Transparency of interfacing with content is enhanced providing an

immediate and direct link via meaningful mapping is in place, thus the user has direct and immediate feedback to input and content response becomes interface affordance (Gibson 1979).

Multimodal fusion[\[edit\]](#)

The process of integrating information from various input modalities and combining them into a complete command is referred to as multimodal fusion.^[6] In literature, three main approaches to the fusion process have been proposed, according to the main architectural levels (recognition and decision) at which the fusion of the input signals can be performed: recognition-based,^{[10][11][32]} decision-based,^{[8][9][12][33][34][35][36]} and hybrid multi-level fusion.^{[5][7][37][38][39][40][41][42]}

The recognition-based fusion (also known as early fusion) consists in merging the outcomes of each modal recognizer by using integration mechanisms, such as, for example, statistical integration techniques, agent theory, hidden Markov models, artificial neural networks, etc. Examples of recognition-based fusion strategies are action frame,^[32] input vectors^[10] and slots.^[11]

The decision-based fusion (also known as late fusion) merges the semantic information that are extracted by using specific dialogue-driven fusion procedures to yield the complete interpretation. Examples of decision-based fusion strategies are typed feature structures,^{[33][38]} melting pots,^{[35][36]} semantic frames,^{[8][12]} and time-stamped lattices.^[9]

The potential applications for multimodal fusion include learning environments, consumer relations, security/surveillance, computer animation, etc. Individually, modes are easily defined, but difficulty arises in having technology consider them a combined fusion.^[43] It's difficult for the algorithms to factor in dimensionality; there exist variables outside of current computation abilities. For example, semantic meaning: two sentences could have the same lexical meaning but different emotional information.^[43]

In the hybrid multi-level fusion, the integration of input modalities is distributed among the recognition and decision levels. The hybrid multi-level fusion includes the following three methodologies: finite-

state transducers,^[38] multimodal grammars^{[7][37][39][40][41][42][44]} and dialogue moves.^[45]

Ambiguity^[edit]

User's actions or commands produce multimodal inputs (multimodal message^[41]), which have to be interpreted by the system. The multimodal message is the medium that enables communication between users and multimodal systems. It is obtained by merging information that are conveyed via several modalities by considering the different types of cooperation between several modalities,^[46] the time relationships^[47] among the involved modalities and the relationships between chunks of information connected with these modalities.^[48]

The natural mapping between the multimodal input, which is provided by several interaction modalities (visual and auditory channel and sense of touch), and information and tasks imply to manage the typical problems of human-human communication, such as ambiguity. An ambiguity arises when more than one interpretation of input is possible. A multimodal ambiguity^[13] arises both, if an element, which is provided by one modality, has more than one interpretation (i.e. ambiguities are propagated at the multimodal level), and/or if elements, connected with each modality, are univocally interpreted, but information referred to different modalities are incoherent at the syntactic or the semantic level (i.e. a multimodal sentence having different meanings or different syntactic structure).

In "The Management of Ambiguities",^[15] the methods for solving ambiguities and for providing the correct interpretation of the user's input are organized in three main classes: prevention, a-posterior resolution and approximation resolution methods.^{[14][16]}

Prevention methods impose users to follow predefined interaction behaviour according to a set of transitions between different allowed states of the interaction process. Example of prevention methods are: procedural method,^[49] reduction of the expressive power of the language grammar,^[50] improvement of the expressive power of the language grammar.^[51]

The a-posterior resolution of ambiguities uses mediation approach.^[17] Examples of mediation techniques are: repetition, e.g. repetition by modality,^[17] granularity of repair^[52] and undo,^[18] and choice.^[19]

The approximation resolution methods do not require any user involvement in the disambiguation process. They can all require the use of some theories, such as [fuzzy logic](#), [Markov random field](#), [Bayesian networks](#) and [hidden Markov models](#).^{[14][16]}

Introduction

Virtual humans have become very popular in the last 15 years, mainly through three-dimensional (3D) movies and games. In movies, they now have very realistic physical and emotional characteristics, including their facial expressions, hair, clothes and motions. In parallel, robotics has made considerable progress and new kinds of robots are available as social companions. Some of them have the appearance of humans. They are physically present, and in case of social robots, we can define them as artificial characters. These social robots share a lot of research in common with Virtual Humans. The high level research on awareness of a situation and the proper reaction to this situation can be addressed by both domains (Virtual Humans and Social robotics).

It is now possible, for example, to replace a real participant in a meeting or an event by its virtual or robotic counterpart. The virtual or robotic counterpart is supposed to give the illusion that the real person is present. This implies that they should look the same as the real human, speaks with the same intonation, and be aware of the real situation, the real participants, and the task currently performed. The virtual human and the robot should react at the right time based on the perception they have from the real participants. It implies to evaluate what each real participant is doing. Perception will be obtained by visual and audio input and recognition. The virtual participant reacts according to the input and its current knowledge. Its reactions encompass animation (body and facial gestures) and speech synthesis.

To tackle the issues on global awareness and proper reactions to real world given situation, the research is focused on three main areas:

- **Tracking of Gestures and Analysis of Real Humans' Behaviour:** This is important as it will give the virtual human and the social robot the capacity to sense and understand what is happening in their environment.
- **Facial and Body Modelling and Animation:** The virtual human or the social robot will need to react and respond appropriately considering the perception of the current situation and the users' state. They should use the same modalities as human does: verbal, body and hand gestures, facial expressions and gaze.

- **Modelling Human Behaviours:** It consists first in modelling personality, mood, and emotion in order to generate the appropriate behaviour; more complex situations correspond to multiple virtual humans interactions and multi-modal and multi-party social interactions.

This book is structured according to these three main research areas and we briefly now summarise the three parts and the corresponding chapters. Part I is dedicated to the analysis and recognition of the situation in order to make the Virtual Human or the Social Robot aware of this situation. First, Chap. 1 addresses the problem of face recognition and facial expression analysis as they are essential abilities of the human and provide the basic visual clues during human–computer interaction. In Chap. 2, it is explained how a social robot or a Virtual Human can understand the meaning of human upper body gestures and express itself by using a combination of body movements, facial expressions and verbal language. It is important to enable the virtual human/social robot such capabilities in order to achieve autonomous behaviour. The human–robot interaction system is based on a novel combination of sensors: the CyberGlove II is employed to capture the hand posture and this feature is combined with the head and arm posture information captured from the Microsoft Kinect. Based on the body posture data, an effective and real-time human gesture recognition method is proposed. Chapter 3 describes how sound source localization and tracking plays an important role in this global analysis of a situation and can support awareness for the virtual human or the social robot. Given the location of a sound, the social robot can be endowed with the capability of sound event awareness, which results in enhanced interaction with human beings.

Part II emphasises the methods for modelling and animating faces and bodies in order to generate the right motion. Chapter 4 is dedicated to conversation, which is clearly important in many applications. It describes the unsolved problem of how the non-verbal component of a conversation might be visualised in a concise yet effective manner that would be suitable for use in a communication skill training scenario. Functionally, conversation serves to deliver and exchange information. However, there is much of a conversation that lays outside of its verbal content yet impacts directly on those involved and in a manner that might be to their detriment or benefit. In Chap. 5, we show that combining 2D images and the range scanned measurement can lead to successful reconstruction results. The

quality shape and collective knowledge from scanned dataset has been exploited to efficiently complement the geometric shape recovery from image inputs. More specifically, a set of 3D body scans that are put in correspondence have been used to parameterise the shape space, which we explore in order to find the optimising parameters that best fit the given image data. The aim of Chap. 6 is to give a comprehensive overview of current state-of-the-art parametric methods for realistic facial modelling and animation. Facial modelling is a fundamental technique in a variety of applications in computer graphics, computer vision and pattern recognition areas. As the 3D sensing technologies have been evolved, the quality of facial modelling has been greatly improved. To enhance the modelling quality and controllability of the

model further, parametric methods, which represent or manipulate facial attributes (e.g., identity, expression, viseme) with a set of control parameters, have been proposed in recent years. Chapter 7 introduces several learning approaches to generate non-preprogrammed motions for a virtual human. To generate non-preprogrammed motions, a Virtual Human should possess the abilities to:

(1) segment a whole movement into meaning segments; (2) learn motion primitives for their adaptation in a changing environment; (3) represent a combination of a motion primitive and its causalities by considering reusability; and finally,

(4) swiftly and reasonably select a dependable motion primitive in accordance with current and goal situations. Chapter 8 presents how to generate natural behaviours of Virtual Humans responsive to the physical interaction with the user such as push and pull. These physical interactions play an important role for increasing the level of immersion of the user and lay foundations for more advanced level of interactions. One of the key components for physical interaction is the generation of suitable balancing behaviours of humanoids against user inputs. In Chap. 9, we introduce the concept of shared object manipulation between real and virtual humans. These can be applied to 3D telepresence applications such as computer-aided design and virtual simulation and training. It consists of three different and complex research domains: (1) virtual object grasping methods for intuitive behaviour (2) virtual object manipulation by virtual humans. (3) consistent management of shared objects to avoid conflicts from multiple simultaneous inputs. The goal of Part III is to present all behavioural aspects of virtual humans and social robots, when they react to real people or to other virtual humans and social robots. Chapter 10 considers affect dynamics which simulates the relation among the emotions, mood and personality and updates emotional states for long-term human-agent interactions. We first examine basic psychological concepts and computational models for affect dynamics. Then we present a psychologically plausible affect dynamics algorithm in which the personality influences the updating of emotional states during the whole interactions. This makes affect dynamics characterised by the personality, which is shown to be important for long-term interactions. Chapter 11 is looking at motion control for the physically realistic robot

Nadine. Robot controllers for such robot need to produce behaviours that match the physical realism of the robot. This chapter describes a robot controller that allows such a robot to fully use the same modalities as humans during interaction including speech, facial and bodily expressions. Chapter 12 presents a review about interactions between real and multiple virtual humans with a focus on virtual assistants and social phobia examples. Interactions between virtual humans are then addressed; particularly gaze attention of other characters and navigation interactions between multiple virtual humans. Chapter 13 focused on interaction of virtual characters and robots interacting with people in social contexts with emphasis on multi-modal and multi-party interactions. The challenges in this area are the estimation of high level user states fusing low level multi-modal sensory input, taking socially appropriate decisions using this partial sensory information and rendering synchronised and timely multi-modal behaviours based on taken decisions.

Part
I User Understanding
Through Multisensory
Perception

Chapter 1

Face and Facial Expressions Recognition and Analysis

Jianfeng Ren, Xudong Jiang and Junsong Yuan

Abstract Face recognition and facial expression analysis are essential abilities of humans, which provide the basic visual clues during human-computer interaction. It is important to enable the virtual human/social robot such capabilities in order to achieve autonomous behavior. Local binary pattern (LBP) has been widely used in face recognition and facial expression analysis. It is popular because of robustness to illumination variation and alignment error. However, local binary pattern still has some limitations, e.g. it is sensitive to image noise. Local ternary pattern (LTP), fuzzy LBP and many other LBP variants partially solve this problem. However, these approaches treat the corrupted image patterns as they are, and do not have an mechanism to recover the underlying patterns. In view of this, we develop a noise-resistant LBP to preserve the image micro-structures in presence of noise. We encode the small pixel difference as an *uncertain* state first, and then determine its value based on the other bits of the LBP code. Most image micro-structures are represented by uniform codes and non-uniform codes mainly represent noise patterns. Therefore, we assign the value of *uncertain* bit so as to form possible uniform codes. In such a way, we develop an error-correction mechanism to recover the distorted image patterns. In addition, we find that some image patterns such as lines are not captured in uniform codes. They represent a set of important local primitives for pattern recognition. We thus define an extended noise-resistant LBP (ENRLBP) to capture line patterns. NRLBP and ENRLBP are validated extensively on face recognition, facial expression analysis and other recognition tasks. They are shown more resistant to image noise compared with LBP, LTP and many other variants. These two

approaches greatly enhance the performance of face recognition and facial expression analysis.

J. Ren (**B**) X. Jiang J. Yuan

BeingThere Centre, Nanyang Technological University, Singapore, Singapore

e-mail: JFREN@ntu.edu.sg

X. Jiang

e-mail: EXDJiang@ntu.edu.sg

J. Yuan

e-mail: JSYUAN@ntu.edu.sg

© Springer International Publishing Switzerland 2016

3

N. Magnenat-Thalmann et al. (eds.), *Context Aware Human-Robot and Human-Agent Interaction*, Human-Computer Interaction Series, DOI 10.1007/978-3-319-19947-4_1

1.1 Introduction

Humans can easily recognize the identity of other people through face images. In fact, face recognition provides an important visual clue for human–computer interaction. It enables the virtual human/social robot to have essential optical capabilities, i.e., to recognize who it is talking to. Then the virtual human/social robot can associate the identity with other information stored in its memory, and enable other high-level interactions. Face recognition is built into the human system as a functional module that provides the user’s identity.

In addition to recognizing faces, humans also easily recognize the facial expressions of others. By analyzing facial expressions, we can understand the emotional mood of people and act accordingly. Thus, it is desirable to enable the virtual human/social robots to recognize the facial expressions of a user, determine the emotional state, and act accordingly to achieve autonomous behavior. Facial expression recognition is thus built into our system as a functional module to analyze the user’s emotion.

Traditionally, holistic features were used in visual recognition, which were derived by vectorizing raw-image pixels. However, the performance of these approaches is often limited by rigorous image alignment and high computational cost. Recently, LBP has become one of the most popular feature representations in visual recognition. The LBP operator transforms an image into an array or image of integer labels describing a micropattern, i.e., a pattern formed by a pixel and its immediate neighbors [32]. More specifically, LBP encodes the signs of the pixel differences between a pixel and its neighboring pixels to a binary code. The histogram of such codes in an image block is commonly used for further analysis. The popularity of this method is due to the following advantages. First, the exact intensities are discarded, and only the relative intensities with respect to the center are preserved; thus, LBP is less sensitive to illumination variations. Second, by extracting the histogram of micropatterns in a patch, the exact location information is discarded and only the patchwise location information is preserved. Thus, LBP is robust to alignment error. Lastly, LBP features can be extracted efficiently, which enables real-time image analysis.

Despite its success, the sensitivity to image noise limits the performance of LBP features [40]. Typically in a smooth image region, a small image

variation may change an LBP bit from 0 to 1, or vice versa, and hence alter the LBP code significantly. In [31], uniform LBP was proposed to reduce the noise in the LBP histogram. Most LBPs in natural images are uniform patterns [1, 31]. Nonuniform patterns are statistically insignificant, and hence noise-prone. By grouping the nonuniform patterns into one label, the noise in nonuniform patterns is suppressed. In [5, 25, 28, 43, 50], information in nonuniform patterns is extracted and used for classification. “Soft histogram” [2, 16, 39] is another approach to improve robustness to noise. Instead of hard-coding the pixel difference, a probability measure is used to represent its likelihood as 0 or 1. However, the probability is closely related to the magnitude of the pixel difference, thus, it is still sensitive to noise. Local ternary pattern (LTP) was

proposed in [40] to tackle the image noise in uniform regions. Instead of binary code, the pixel difference is encoded as a 3-valued code according to a threshold t . Then, the ternary code is split into a positive LBP and a negative LBP in order to reduce the dimensionality. LTP showed to be less sensitive to noise, especially in uniform regions [40]. Subsequently, many LTP variants [3, 10, 13, 21, 22, 27] were proposed in the literature. LBP and its variants partially solve the noise-sensitive problem; however, they lack the mechanism to recover corrupted image patterns. In this chapter, we propose a noise-resistant LBP (NRLBP) and an extended noise-resistant LBP (ENRLBP) to address this issue.

The signs of pixel differences used to compute LBP and its variants are vulnerable to noise when they are small. Thus, we propose to encode small pixel difference as an *uncertain* bit first and then determine its value based on the other bits of the LBP code. Uniform patterns are more likely to occur as against nonuniform patterns in natural images [1, 31]. Most image structures are represented by uniform patterns, while nonuniform patterns are most likely caused by noise. Thus, in NRLBP, the values of *uncertain* bits are assigned to form uniform patterns. A nonuniform pattern is generated only if no uniform pattern can be formed. As noise may change a uniform pattern into an unstable nonuniform pattern, NRLBP corrects many distorted nonuniform patterns to uniform patterns.

For LBP and LTP, line patterns are treated as nonuniform patterns and grouped into the nonuniform bin. Uniform patterns mainly represent spot, flat region, edge, edge end, and corner. A local image is a line pattern if it is a line against the background, as shown in Fig. 1.6. Line patterns may appear less frequently than uniform patterns, but they represent an important group of local primitives for pattern recognition. Thus, we propose an extended set of uniform patterns corresponding to line patterns. Then, we propose extended noise-resistant LBP (ENRLBP). During the encoding process, we assign the values of *uncertain* bits so as to form extended uniform patterns.

To evaluate NRLBP, we first inject Gaussian noise and uniform noise of different levels on the AR database [24] for face recognition and facial expression recognition. NRLBP and ENRLBP demonstrate strong resistance to noise compared with LBP/LTP and its variants. They are further compared with LBP/LTP variants for face recognition on the

extended Yale database [8, 18] and the O2FN database [33]. NRLBP and ENRLBP consistently achieve better performance.

1.2 Literature Review

LBP and its variants have been widely used in many recognition applications, e.g., facial analysis [1, 34, 37, 40], texture classification [9, 20, 25, 31], dynamic texture

recognition [35, 47, 49], human detection [26, 42], scene classification [36, 43, 44] and others [14, 22, 38]. LBP encodes the relative pixel difference of the neighboring pixels to the central pixels, e.g., $LBP_{P,R}$ denotes the LBP code of P neighbors at a distance of R . Denote the gray level of the center pixel by i_c , and the gray level of the circular neighbors by i_p , where $p = 0, 1, \dots, P - 1$. The LBP code for pixel (x, y)

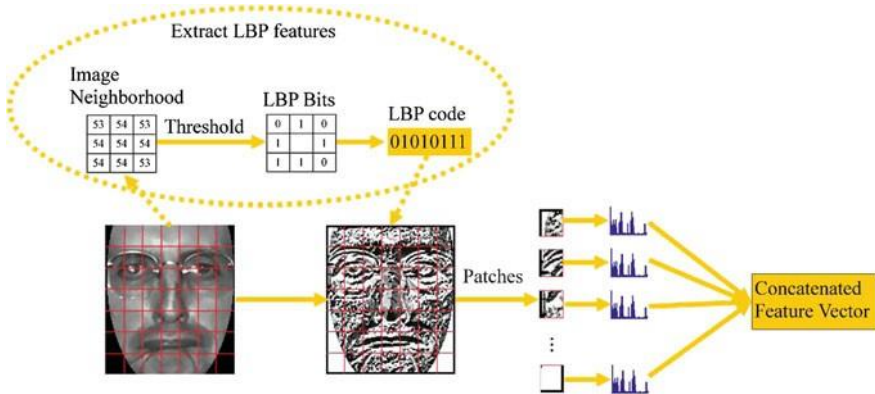


Fig. 1.1 Block diagram of LBP encoding process

is obtained
as

$$LBP_{p,R}(x, y) = \sum_{p=0}^{p-1} s(i_p - i_c) 2^p, \quad (1.1)$$

where $s(z)$ is the threshold function

$$s(z) = \begin{cases} 1 & \text{if } z \geq 0, \\ 0 & \text{if } z < 0. \end{cases} \quad (1.2)$$

The histogram of these codes is then used for further analysis. The block diagram of LBP encoding process is shown in Fig. 1.1. The image is first encoded as an LBP image that is divided into patches, and one LBP histogram is extracted from each patch. Finally, the LBP histograms of all patches are concatenated to form the final feature vector.

Despite its wide applications, the LBP feature still has some limitations, i.e., it is sensitive to image noise. A small image variation will alter the LBP bit from 0 to 1 or vice versa, and hence alter the code significantly. Many

approaches have been proposed to address this issue, which are grouped into the following four categories.

1.2.1 Uniform LBP

It is shown in [1, 31] that most LBP codes in a natural image are uniform codes. An LBP code is defined as a uniform pattern if it has at most two circularly bit- wise transitions from 0 to 1 or vice versa, and nonuniform patterns if otherwise. For example, “11000000” is a uniform code, while “11001100” is a nonuniform code.

For $LBP_{8,2}$, there are 58 uniform codes and 198 nonuniform codes. Uniform patterns are statistically more significant and their occurrence probabilities can be more reliably estimated. By contrast, nonuniform patterns are statistically less insignificant, and hence noise-prone and unreliable. In uniform LBP mapping [31], one separate histogram bin is used for each uniform pattern and all nonuniform patterns are accumulated in a single bin. By grouping the nonuniform patterns into one label, the noise in nonuniform patterns is suppressed, while at the same time the number of patterns is reduced significantly.

In [5, 25, 28, 43, 50], information in nonuniform patterns is extracted and also used for classification. Liao et al. proposed a dominant LBP (DLBP) that considers the most frequently occurred patterns in a texture image [25]. This is based on the observation that some nonuniform codes may have higher occurrence frequency than uniform codes. Instead of being based on image microstructures [31], DLBP chooses LBP codes based on occurrence frequency directly. Zhou et al. [50] and Fathi et al. [5] proposed to extract information from nonuniform patterns based on pattern uniformity measure and the number of 1s in the LBP codes. Principal component analysis [43] and random subspace approach [28] were used to extract information from the whole LBP histogram, including both uniform and nonuniform patterns. These approaches can extract discriminant information from the nonuniform codes. However, as the nonuniform codes may contain noise, these approaches tend to be sensitive to noise.

1.2.2 Fuzzy LBP

“Soft histogram” is another approach to improve the robustness to noise. Instead of hard-coding the pixel difference as shown in Eq. (1.2), a probability measure is used to represent its likelihood as 0 or 1. In [2, 16], a fuzzy LBP (FLBP) using piecewise linear fuzzy membership function was proposed that replaces $s(z)$ with two fuzzy membership functions:

$$f^1(z, d)$$

$$=$$

$$f^1(z, d) = \begin{cases} 0 & \text{if } z < -d, \\ 0.5 + 0.5z/d & \text{if } -d \leq z \leq d, \\ 1 & \text{if } z > d. \end{cases} \quad (1.3)$$

$$f^0(z, d) = 1 - f^1(z, d). \quad (1.4)$$

The parameter d controls the amount of fuzzification. $f^1(z, d)$ and $f^0(z, d)$ represent the probability that the pixel difference z is encoded as 1 and 0, respectively. When constructing the LBP histogram, we calculate the probabilities of all patterns as

$$P_j = \sum_{i=0}^{P-1} c_i f^1(z_i, d) + (1 - c_i) f^0(z_i, d), \quad (1.5)$$

where $j = \sum_{i=0}^{P-1} c_i \times 2^i$ and c_i is i th bit of the code. For example, the probability of “11001100” is calculated as $P_{11001100} = f_7^1 f_6^1 f_5^0 f_4^0 f_3^1 f_2^1 f_1^0 f_0^0$, where f_i^b is the probability of bit i to be encoded as $b \in \{0, 1\}$. The complete histogram is derived by

summing the contributions of all pixels in an image patch.

Another fuzzy LBP uses Gaussian-like membership function [39], which is defined as

$$f^1(x, \sigma) = \begin{cases} 1 - 0.5e^{-x^2/\sigma^2} & \text{if } x \geq 0, \\ 0.5e^{-x^2/\sigma^2} & \text{if } x < 0, \end{cases} \quad (1.6)$$

$$f^0(x, \sigma) = 1 - f^1(x, \sigma), \quad (1.7)$$

where $x = (i_p - i_c)/i_c$ and σ controls the amount of fuzzification. A comprehensive comparison between LBP and fuzzy LBP in classifying and segmenting textures can be found in [17].

When the original LBP operator is used, one pixel only contributes to one histogram bin. In fuzzy LBP, one pixel typically contributes to more than one bin. The contributions of a pixel to all bins always sum to 1. Using fuzzy LBP, a small change in the input image causes only a small change in the histogram. Thus, it is robust to image noise to a certain extent. However, the probability measures defined in Eqs. (1.3) and (1.6) are closely related to the magnitude of the pixel difference. Thus, it is still sensitive to noise.

1.2.3 LTP and Its Variants

The LBP code is sensitive to noise, especially in a near-constant image region. In LTP [40], the pixel difference z between a pixel and its neighbors is encoded by three values (1, 0 or -1) according to a threshold t , i.e.,

$$s_T(z) = \begin{cases} 1 & \text{if } z > t, \\ 0 & \text{if } -t \leq z \leq t, \\ -1 & \text{if } z < -t. \end{cases} \quad (1.8)$$

State “0” is used to handle the small image variations, i.e., if the difference is within the threshold t , it is encoded as state “0”. These small variations are treated separately. Thus, LTP is less sensitive to noise. However, the LTP histogram is of high dimensionality, e.g., $LTP_{8,2}$ will exhibit a histogram of $3^8 = 6561$ bins. To reduce the dimensionality, the ternary pattern is split into a positive LBP code and a negative LBP code in [40].

Subsequently, many LTP variants were proposed in the literature. Nanni et al. proposed a quinary code of five values according to two thresholds [27]:

$$s_q(z) = \begin{cases} 2 & \text{if } z \geq t_2, \\ 1 & \text{if } t_1 \leq z < t_2, \\ 0 & \text{if } -t_1 \leq z < t_1, \\ -1 & \text{if } -t_2 \leq z < -t_1, \\ -2 & \text{if } z < -t_2, \end{cases} \quad (1.9)$$

where t_1 , t_2 are two thresholds and $t_1 < t_2$. Then the quinary code is split into four binary codes, analogous to LTP. As LTP is not invariant under scaling of intensity values, Liao et al. proposed scale-invariant local ternary pattern (SILTP) to deal with the gray scale intensity changes in a complex background [22]. To reduce the high dimensionality of LTP, center-symmetric LTP (CSLTP) was proposed in [10]. Instead of using the pixel differences between the neighboring pixels and the central pixel, the pixel differences between diagonal neighbors are calculated. In local adaptive ternary patterns (LATP) [3] and extended LTP (ELTP) [21], instead of using a constant threshold, the threshold is calculated for each window using local statistics, which makes them less sensitive to illumination variations. In local triplet pattern [13], the equality is modeled as a separate state, and a tri-state pattern is formulated. It can be viewed as a special case of LTP [40].

To improve the robustness to noise, other thresholding and encoding techniques have been proposed. Instead of comparing with the center pixel, Hafiane et al. proposed median binary pattern (MBP) by thresholding the local pixel values against the median value within the local neighborhood [11]. In improved LBP [6], the neighboring pixels are thresholded against the mean intensity of the local neighborhood. Trefny and Matas [41] proposed two new encoding schemes: binary value transition coded LBP (tLBP) using the pixel differences of neighbors in clockwise direction and direction coded LBP (dLBP) encoding the second-order pixel differences. In centralized binary pattern (CBP) [7], the average intensity within the neighborhood is compared to the center pixel and added as an LBP bit. To reduce the sensitivity to noise, a Bayesian LBP (BLBP) was developed by He et al. [12]. It is formulated in a filtering, labeling, and

statistic (FLS) framework, in which the labeling procedure is modeled as an optimization process.

1.2.4 *Preprocessing*

Image preprocessing techniques can also improve the robustness of LBP to image noise. Gabor filtering is often used before extracting LBP feature, as LBP captures image microstructures and Gabor filters encode image information on a large scale. Zhang et al. [46] proposed to extract LBP features from 40 Gabor-filtered images, known as local gabor binary patterns (LGBP). It has been used as a benchmark

method in many recent face recognition studies, whose drawback however is its high dimensionality. Tan and Triggs [40] proposed an image preprocessing framework that consists of gamma correction, difference of Gaussian (DoG) filtering, masking (optional), and equalization of variations. Used with LTP, it has demonstrated superior performance in face recognition.

To tackle the small image variations in a near-constant region, Yao and Chen [45] proposed local edge patterns (LEP) for color texture retrieval, in which the Sobel edge detection is used to suppress the small image variations in the near-constant region, and LBP-like feature is extracted on the filtered image. Similar ideas have been used in [15] for shape localization and in [48] for facial image representation. Li et al. [19] proposed to extract LBP features from multiscale heat kernels, which capture the intrinsic topological structural information of face appearance. In [23], LBP features are extracted from curvelet-transformed images for medical image analysis.

1.2.5 Discussion

Although previous approaches partially solve the noise-sensitivity problem, they treat corrupted LBP patterns as they are, and lack the mechanism to recover the underlying patterns. In the following section, we present our noise-resistant LBP with an embedded error-correction mechanism that better handles image noise.

1.3 Noise-Resistant Local Binary Patterns

1.3.1 Problem Analysis of LBP and LTP

LBP encodes the pixel difference $z_p = i_p - i_c$ between the neighboring pixel i_p

and the central pixel i_c . Let $C^B = \overbrace{b^B \ b^B \ \dots \ b^B}^P$ denote the LBP code of P

$$P, R \quad P-1 \ P-2 \quad 1 \ 0$$

neighbors at distance R to the center pixel. A code is also called a pattern. Let $LBP_{P,R}$

denote such a coding scheme for C^B . Each bit is obtained as in Eq. (1.2).

LBP is widely used in many applications because of its simplicity and robustness to illumination variations; its drawback is it is sensitive to image noise. In [31], uniform LBP was proposed to capture fundamental image structures and reduce the noise in an LBP histogram. The uniformity U is defined as the number of circularly

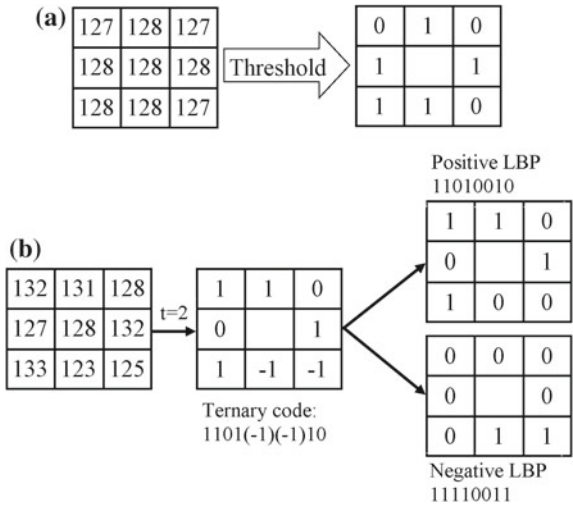
bitwise transitions from 0 to 1 or vice versa. An LBP is $u2$ -uniform or simply called uniform if $U \leq 2$. For example, "11110000" is a uniform pattern as $U = 2$, while

"01010111" shown in Fig. 1.2a is a nonuniform pattern as $U = 6$, LBP^{u2} indicates

a coding and histogram mapping scheme in which $u2$ -uniform LBP codes of P neighbors at the distance of R to the center pixel are used. Uniform patterns occur much more frequently than nonuniform patterns in natural images. It has been shown

Fig. 1.2 a An example of LBP encoding scheme for

the smooth region with small image noise. LBP is sensitive to image noise. **b** An example of LTP encoding process. LTP doubles the number of patterns compared with LBP



that $LBP_{8,1}^{u2}$ accounts for almost 90 % of all patterns for texture images [31] while

$LBP_{8,2}^{u2}$ accounts for 90.6 % for facial images [1]. The occurrence probabilities of

nonuniform patterns are so small that they cannot be reliably estimated [31]. Inclusion

of such noisy estimates in the histogram would harm the classification performance. In addition, nonuniform patterns may be caused by the image noise. Therefore, when constructing the histogram, all nonuniform patterns are grouped into one bin. This not only reduces feature dimensionality, but more importantly, the noise due to unreliable

estimates of nonuniform patterns is greatly suppressed. The number of patterns is reduced significantly from 2^P to $P(P - 1) + 3$. For example, $LBP_{8,2}$ consists of 256

patterns, while $LBP_{8,2}^{u2}$ has only 59 patterns.

Uniform LBP successfully reduces noise in the LBP histogram, though it is still sensitive to image noise. As shown in Fig. 1.2a, a small noise will cause the pixel difference to be encoded differently. Ideally such a smooth region should be encoded as “11111111,” but owing to image noise it is encoded as “01010111” instead. LTP partially solves this problem by encoding the small pixel difference into a third state [40]. Instead of using binary code, each pixel difference is encoded as a 3-valued code. Let $C^T = \overrightarrow{b^T \ b^T \ \cdots \ b^T \ b^T}$ denote the LTP code of P neighbors

$$C^T = \overrightarrow{b_{P,R}^T \ b_{P-1,R}^T \ b_{P-2,R}^T \ \cdots \ b_{1,R}^T \ b_{0,R}^T} \quad (1.9)$$

at distance R to the center pixel and $LTP_{P,R}$ denote such a coding scheme for $C_{P,R}$. Each bit is obtained as

$$b_p^T = \begin{cases} 1 & \text{if } z_p \geq t, \\ 0 & \text{if } |z_p| < t_p, \\ -1 & \text{if } z_p \leq -t, \end{cases} \quad (1.10)$$

where t is a predefined threshold.

LTP is more resistant to noise. However, the dimensionality of LTP histogram is very large, e.g., $LTP_{8,2}$ exhibits a histogram of $3^8 = 6561$ bins. Thus, in [40], LTP is split into a positive LBP and a negative LBP. Each bit of positive LBP is obtained as

$$b_p^p = \begin{cases} 1 & \text{if } z_p \geq t, \\ 0 & \text{if } z_p < t. \end{cases} \quad (1.11)$$

Each bit of negative LBP is obtained as

$$b_p^n = \begin{cases} 0 & \text{if } z_p \leq -t, \\ 1 & \text{if } z_p > -t. \end{cases} \quad (1.12)$$

To show the similarities and differences among LBP, LTP, and NRLBP clearly, the negative LBP defined here is the complement of the negative LBP defined in [40]. Effectively, they achieve the same result for histogram-based comparison. Eventually, LTP is treated as two separate channels of LBP codes: one channel for positive LBP and the other for negative LBP. In general, uniform LTP is used in which both channels are uniform LBP. This coding scheme is denoted as LTP^{u2} . An example of LTP encoding process is shown in Fig. 1.2b. LTP doubles the number of patterns compared to LBP.

The small pixel difference may be easily distorted by noise. Both LBP and LTP lack the mechanism to correct the corrupted patterns. The corrupted image patterns are treated without any attempt to recover the underlining local structures. To address this issue, we propose a noise-resistant LBP and an extended noise-resistant LBP.

1.3.2 Noise-Resistant LBP

LBP is sensitive to noise. Even a small noise may change the LBP code significantly. Thus, we propose to encode the small pixel difference as an *uncertain* bit X first and then determine X based on other certain bits of the LBP code. As regards the pixel difference z_p between the neighboring pixel and the central pixel,⁹ we encode it into one of the three states b^N as

$$b_p^N = \begin{cases} 1 & \text{if } z_p \geq t, \\ X & \text{if } |z_p| < t_p, \\ 0 & \text{if } z_p \leq -t. \end{cases} \quad (1.13)$$

States 1 and 0 represent two strong states where the pixel difference is almost definitely positive and negative, respectively. Noise can unlikely change them from 0 to 1 or from 1 to 0 . State X represents an *uncertain* state where the pixel difference

is small. A small pixel difference is vulnerable to noise if we only take its sign. More specifically, noise can easily change its LBP bit from 0 to 1 or vice versa. Therefore, we encode it as an *uncertain* state regardless of its sign.

Then, we constrain the value of the *uncertain* bit into either 0 or 1, represented by a variable x_i , $x_i \in \{0, 1\}$. Let $\mathbf{X} = (x_1, x_2, \dots, x_n)$ denote the vector formed by n

variables of a code. $\mathbf{X} \in \{0, 1\}^n$. The *uncertain* code can be represented by $C(\mathbf{X})$ as

$$\overrightarrow{b^N \quad b^N \quad \dots \quad b^N b^N} = C(\mathbf{X}). \quad (1.14)$$

$P-1 \quad P-2 \qquad \qquad 1 \quad 0$

Take the *uncertain* code “11X100X0” in Fig. 1.3a as illustration. The *uncertain* code $\overrightarrow{11x_2100x_10}$ can be viewed as a function of $\mathbf{X} = \{x_1, x_2, \dots\}$

After we derive the *uncertain* code, we determine the *uncertain* bits based on the

values of the other certain bits to form one or more codes of image local structures. Uniform patterns represent local primitives, including spot, flat, edge, edge end, and corner. They appear more often than nonuniform patterns in natural images. Since uniform patterns occur more likely than nonuniform ones, we assign the values of *uncertain* bits $\mathbf{X}$ so as to form possible uniform LBP codes. A nonuniform pattern is generated only if no uniform pattern can be formed. Take Fig. 1.3b as an example. We determine the *uncertain* bit of *uncertain* code “11X1X0X0” so as to form only uniform patterns, e.g., “11110000” and “11111000.”

Mathematically, let Φ_u denote the collection of all uniform LBP codes. For LBP^{u2} ,

Φ_u consists of 58 uniform codes. Based on the *uncertain* code $C(\mathbf{X})$, a set of the NRLBP codes are obtained as

$$S_{NRLBP} = \{C(\mathbf{X}) | \mathbf{X} \in \{0, 1\}^n, C(\mathbf{X}) \in \Phi_u\} \quad (1.15)$$

Now let us construct the histogram of NRLBP for a local image patch. Let m denote the number of elements in S_{NRLBP} . If $m > 0$, the bin corresponding to each element in S_{NRLBP} will be added by $1/m$. After all, all these patterns originate from one *uncertain* code. If $m = 0$, the nonuniform bin will be added by 1. This process is repeated for every pixel in the patch. Algorithm 1 summarizes the process.

Algorithm 1 Histogram construction for NRLBP

for Every pixel in a patch **do**

1. Derive the *uncertain* code $C(\mathbf{X})$ as in Eqs. (1.13), (1.14).
3. Search *uncertain* bits $\mathbf{X}$ in the space $0\{1^n\}$ so that $C(\mathbf{X})$ forms uniform LBP codes as in Eq. (1.15).
4. Construct the histogram.

if $m = 0$ **then**

 Accumulate the nonuniform bin with 1.

else

 Accumulate the bin of each pattern in S_{NRLBP} with $1/m$.

end if

end for

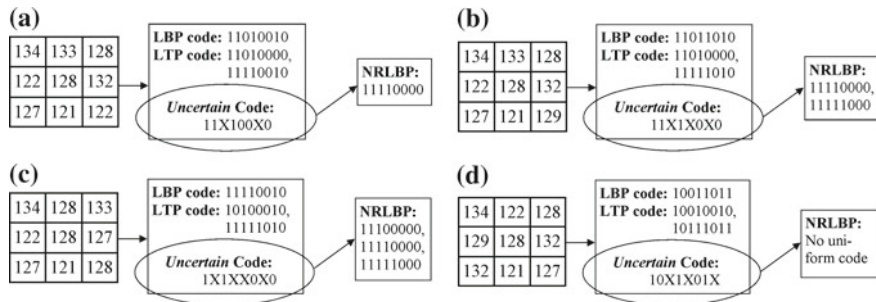


Fig. 1.3 Illustration of encoding process of NRLTP and comparison with LBP and LTP. **a–c** correspond to $m=1, 2, 3$ resulting in NRLBP codes, respectively. **d** shows an example that no uniform code can be formed. NRLBP is significantly different from LBP and LTP. Threshold t is chosen as 2 for LTP and NRLBP in this figure. **a** $m = 1$, **b** $m = 2$, **c** $m = 3$, **d** $m = 0$

Now we compare NRLBP with LBP and LTP with examples. We consider the cases that different number of LBP codes are derived in S_{NRLBP} . Image patterns in Fig. 1.3a–c generate $m = 1, 2, 3$ NRLBP codes, respectively. Figure 1.3d shows an example where no uniform code can be formed for NRLBP. The corresponding LBP and LTP codes are also given. For LTP, the positive and negative LBPs are accumulated in two different histograms, while for LBP and NRLBP, the codes are accumulated in one histogram.

As noise may change a uniform image pattern into an unstable nonuniform pattern, NRLBP corrects such a code back to uniform code. As shown in Fig. 1.3a, the LBP code is “11010010,” which may be distorted by noise. NRLBP first derives the *uncertain* code “11X100X0,” and then determines its *uncertain* bits by forming the uniform code “11110000.” This can be viewed as an error-correction mechanism. Note that we only attempt such an error correction on *uncertain* bits; we do not attempt to correct the nonuniform patterns that result from two strong states. Similarly, we can observe such an error-correction process in Fig. 1.3b, c. In these two cases, more than one NRLBP code is generated.

NRLBP corrects noisy nonuniform patterns back to uniform pattern. Figure 1.4 shows the histograms of LBP, LTP, and NRLBP for the image shown in Fig. 1.8g. The threshold t is chosen as 10 for LTP and NRLBP. The LTP histogram is the concatenation of the positive and negative LBP histograms. The last bin of each histogram corresponds to nonuniform patterns, while other bins correspond to uniform patterns. Clearly, compared with the LBP

and LTP histograms, nonuniform patterns in the NRLBP histogram are reduced significantly from about 35 % to about 10 % only. NRLBP corrects a large amount of nonuniform patterns that are corrupted by noise back to uniform patterns.

NRLBP is different from LBP and LTP in many other aspects, besides the capability of noise resistance and error-correction. The LBP code is that of the NRLBP code set if it is uniform. The only exception is that the LBP code is nonuniform and is corrected back to uniform code in NRLBP. Compared with LTP, the treatment of *uncertain* state is totally different for NRLBP. For LTP, all *uncertain* bits are set to

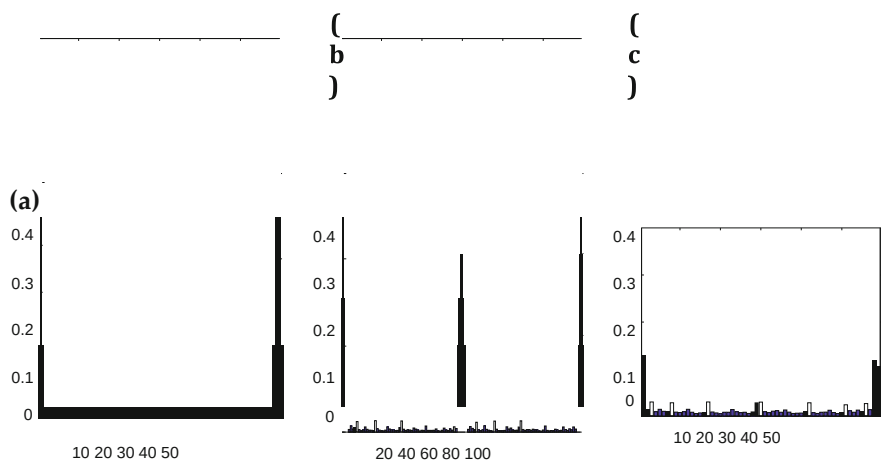


Fig. 1.4 The histograms of LBP, LTP, and NRLBP for the image shown in Fig. 1.8g. The LTP histogram is a concatenation of the positive and negative LBP histograms. The last bin of each histogram corresponds to nonuniform patterns. Compared with the LBP and LTP histograms, NRLBP significantly reduces nonuniform patterns from about 35% to about 10%. NRLBP corrects a large amount of noisy nonuniform patterns back to uniform patterns. **a** LBP histogram. **b** LTP histogram. **c** NRLBP histogram

0 for positive half and 1 for negative half as shown in Fig. 1.3, whereas for NRLBP, we do not hurry for a decision on the *uncertain* bits. We treat them as if they could be encoded as 1 and/or 0, and determine their values based on the other bits of code. Mathematically, for LTP, X_0^n for positive half and X_1^n for negative half, whereas $X_0, 1^n$ for NRLBP. The number of histogram bins is also different. The LTP histogram consists of 118 bins, while the NRLBP histogram has only 59 bins.

For implementation, a lookup table from the *uncertain* code to the feature vector of NRLBP histogram can be precomputed. Then, the feature vector of local image patch can be easily obtained by summing up the feature vector of each pixel in this image patch.

1.3.3 Extended Noise-Resistant LBP

The local primitives represented by uniform LBP mainly consist of spots, flat region, edges, edge ends, and corners [32], as shown in Fig. 1.5. However, a large group of local primitives are totally discarded, e.g., lines patterns, as shown in Fig. 1.6. Although these patterns may not appear as frequently as uniform patterns, they represent an important group of local primitives that may be crucial for recognition tasks. Grouping them with other nonuniform patterns into one bin may result in information loss. Therefore, we introduce an extended set of uniform patterns to preserve line patterns. Of all possible line patterns, diagonal lines appear less frequently. In order to keep the feature vector compact, we only choose nearly horizontal or vertical lines.

Let α denote the angle of the line away from the horizontal line. If $\alpha \in [0, 30^\circ)$ or $\alpha \in (150^\circ, 180^\circ]$, it is considered a horizontal line. If $\alpha \in [60^\circ, 120^\circ]$, it is considered a vertical line. If $\alpha \in [30^\circ, 60^\circ)$ or $\alpha \in (120^\circ, 150^\circ]$, it is considered a

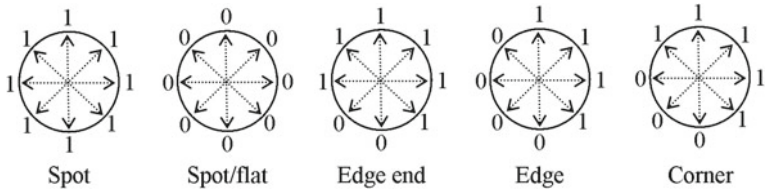


Fig. 1.5 Local primitives detected by $LBP^{u2}_{8,2}$

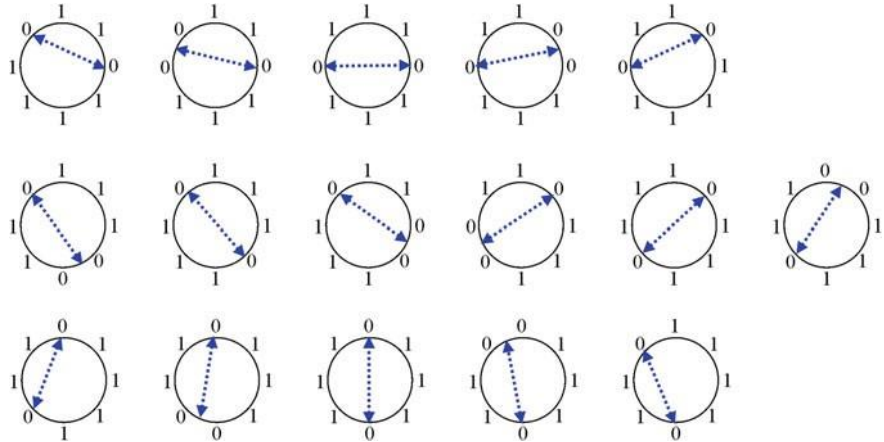


Fig. 1.6 Samples of line patterns. The three rows correspond to *horizontal*, *diagonal*, and *vertical* lines. The *diagonal* lines are rare patterns for natural images and hence are discarded. The remaining *horizontal* and *vertical* lines are the extended set of uniform patterns

diagonal line. Figure 1.6 shows some samples of horizontal, diagonal, and vertical lines.

The extended set of uniform patterns consist of 48 patterns. In addition to 58 uniform patterns, we derive the extended uniform patterns. Analogous to NRLBP, we can derive the extended NRLBP (ENRLBP). Instead of forming uniform patterns we form extended uniform patterns as our ENRLBP pattern. In such a way, line patterns are preserved during the encoding process. The number of bins of ENRLBP histogram is 107, which is smaller than the LTP histogram which has 118 bins.

1.3.4 *Applications on Face and Facial Expression Recognition*

NRLBP and ENRLBP can be used for many recognition tasks. Here, we focus on the applications on face and facial expression recognition. The block diagram for the face recognition system is shown in Fig. [1.7](#).

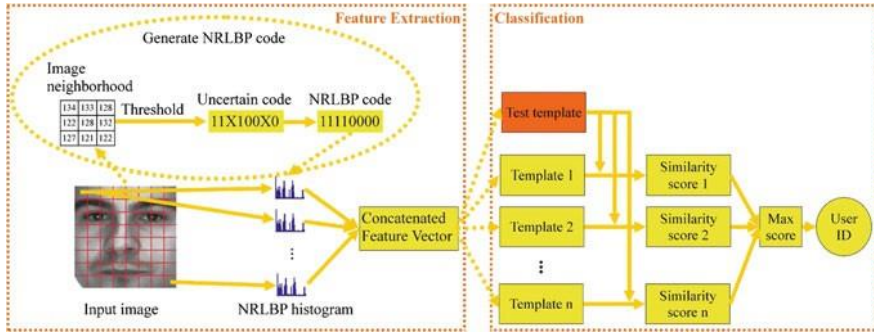


Fig. 1.7 Block diagram for face recognition system

The system consists of two parts: feature extraction and classification. Before feature extraction, we first geometrically normalize an image based on two eye locations so that the two eyes are in the same locations for all images. This typically involves rotation, scaling, and translation of the image. Then we divide the image into patches. One NRLBP histogram is extracted from each patch using Algorithm 1. The histograms of all patches are concatenated to form the final feature vector.

For classification, we use the nearest-neighbor (NN) classifier with three different distance measures: Chi-square distance, histogram intersection distance, and G-statistic as defined in Eqs. (1.16)–(1.18), respectively.

$$\chi^2(\mathbf{x}, \mathbf{y}) = \sum_{i,j} \frac{(x_{i,j} - y_{i,j})^2}{x_{i,j} + y_{i,j}}, \quad (1.16)$$

$$\mathbf{D}_{HI}(\mathbf{x}, \mathbf{y}) = - \sum_{i,j} \min(x_{i,j}, y_{i,j}), \quad (1.17)$$

$$\mathbf{D}_G(\mathbf{x}, \mathbf{y}) = - \sum_{i,j} x_{i,j} \log y_{i,j}, \quad (1.18)$$

where $\mathbf{x}, \mathbf{y}$ are the concatenated LBP feature vectors of two image samples; $x_{i,j}$ and $y_{i,j}$ are j th dimension of i th patch. The G-statistic is numerically unstable, as many histogram bins may have zero elements, which easily causes $D_G \inf$. Thus, we modify it into a numerically stable form

$$D_G(\mathbf{x}, \mathbf{y}) = - \sum_{i,j} x_{i,j} \log(x_{i,j} + y_{i,j}), \quad (1.19)$$

Only when both $x_{i,j}$ and $y_{i,j}$ are zero, we set $\ominus \log(0) = 0$. We call this distance measure as modified G-statistic (MG). MG is numerically more stable and hence can better handle the problem of too few elements in the histogram than G-statistic. For facial expression recognition, we use a similar approach, i.e., the NRLBP features are extracted in exactly the same way as face recognition. For classification, instead of user ID, we label each image with facial expressions. The objective of the classification is to determine which facial expression it is.

1.4 Experimental Results

We conducted extensive experiments to validate the advantages of NRLBP and ENRLBP. Table 1.1 summarizes the approaches compared with the classifiers used and the applications tested. NRLBP and ENRLBP are compared with uniform LBP_{8,2}

and uniform LTP, i.e., $LBP_{8,2}^{u2}$ and $LTP_{8,2}^{u2}$ are used. Let $NRLBP_{P,R}$, $ENRLBP_{P,R}$ and $LBP_{P,R}^{u2}$

denote the coding schemes for NRLBP and ENRLBP using P neighbors at distance R to the center pixel, respectively. The number of features for each patch is 59 for $LBP_{8,2}^{u2}$, 118 for $LTP_{8,2}^{u2}$, 59 for $NRLBP_{8,2}$, and 107 for $ENRLBP_{8,2}$. Domi-

$LBP_{8,2}^{u2}$ $LTP_{8,2}^{u2}$

inant LBP (DLBP) [25], novel extended LBP (NELBP) [50], and noise tolerant LBP (NTLBP) [5] are compared as they extract information from nonuniform bins, analogous to our approaches. We choose the dominant patterns that account for 80 % of the total pattern occurrences, same as in [25]. Fuzzy LBP (FLBP) [2, 16, 17] is also compared. We implement fuzzy LBP using piecewise linear fuzzy membership function in [2].

We conducted comparative experiments for various applications. First, we injected Gaussian and uniform noise of various noise levels onto images of the AR data- base [24] for face recognition and the Outex-13 dataset [30] for texture recognition. NRLBP and ENRLBP are compared with various LBP/LTP variants in order to val-

Table 1.1 Summary of the approaches compared with the classifiers used and the application tested

The approaches	The classifier	The applications
LBP [29]	Nearest-neighbor classifier + Chi-square distance	Facial expression recognition on the AR database [24]
LTP [40]	Nearest-neighbor classifier + histogram intersection distance	Face recognition on the AR database [24]
Dominant LBP [25]	Nearest-neighbor classifier + modified G-statistic	Face recognition on the extended Yale B database [8, 18]
Fuzzy LBP [2, 16, 17]		Face recognition on the O2FN database [33]
Novel extended LBP [50]		
Noise tolerant LBP [5]		

idate their noise-resistance property. Then we apply them to real images that are noise-prone. Illumination variation is one of the challenges in face recognition. We conducted experiments on two challenging face databases with large illumination variations: the extended Yale B database [8, 18] and the O2FN database [33]. They are compared with LBP/LTP variants for protein cellular classification on the 2D Hela database [4] and image segmentation on the image of the Outex segmentation database [30] and one image from the Web. In order to reduce illumination variations, the images of the Outex-13 dataset, the extended Yale B database, and the O2FN database are preprocessed similarly as in [40]. We use the source codes provided by the authors of [40] to perform this photometric normalization.

1.4.1 *Facial Expression Recognition on the AR Database*

On the AR database [24], NRLBP and ENRLBP are compared with LBP/LTP variants on images injected with noise in order to demonstrate their noise-resistant property. The AR database is of high resolution and high image quality, and is considered as a face database with almost no image noise. We choose 75 subjects, where each subject contains images from two sections. Each section contains images of four different facial expressions: neutral, smile, anger, and scream, as shown in Fig. 1.8. We use the images of the first section as the gallery set and the images of the second section as the probe set. Gaussian noise is one of the most common types of noise. We normalize the images in the range of $(0, 1)$, and then apply additive Gaussian noise with zero mean and standard derivation of σ . We conduct experiments for



Fig. 1.8 First row shows images with different facial expressions. Second row shows images with additive Gaussian noise of $\sigma = 0, 0.05, 0.1, 0.15$, respectively. **a** Neutral. **b** Smile. **c** Anger. **d** Scream. **e** $\rho = 0$. **f** $\rho = 0.05$. **g** $\rho = 0.10$. **h** $\rho = 0.15$

Table 1.2 Summary of the average facial expression recognition rate of each approach at optimal setting on the AR database injected with Gaussian noise

Algorithm	χ^2 Distance, $\sigma =$			Histogram intersection, $\sigma =$			Modified G-statistic, $\sigma =$		
	0.05 (%)	0.10 (%)	0.15 (%)	0.05 (%)	0.10 (%)	0.15 (%)	0.05 (%)	0.10 (%)	0.15 (%)
LBP	85.33	76.00	63.67	86.00	74.00	58.00	83.33	69.00	50.67
LTP	87.33	79.33	72.00	86.33	78.00	69.00	86.67	74.00	62.67
DLBP	86.33	79.67	67.67	88.00	78.67	67.67	88.00	79.33	65.00
FLBP	87.33	83.67	79.33	86.33	83.67	79.00	87.00	84.00	79.67
NELBP	79.00	63.67	46.67	78.00	61.67	44.67	79.00	59.67	41.00
NTLBP	80.67	53.67	40.67	80.00	61.67	45.33	79.00	54.33	37.67

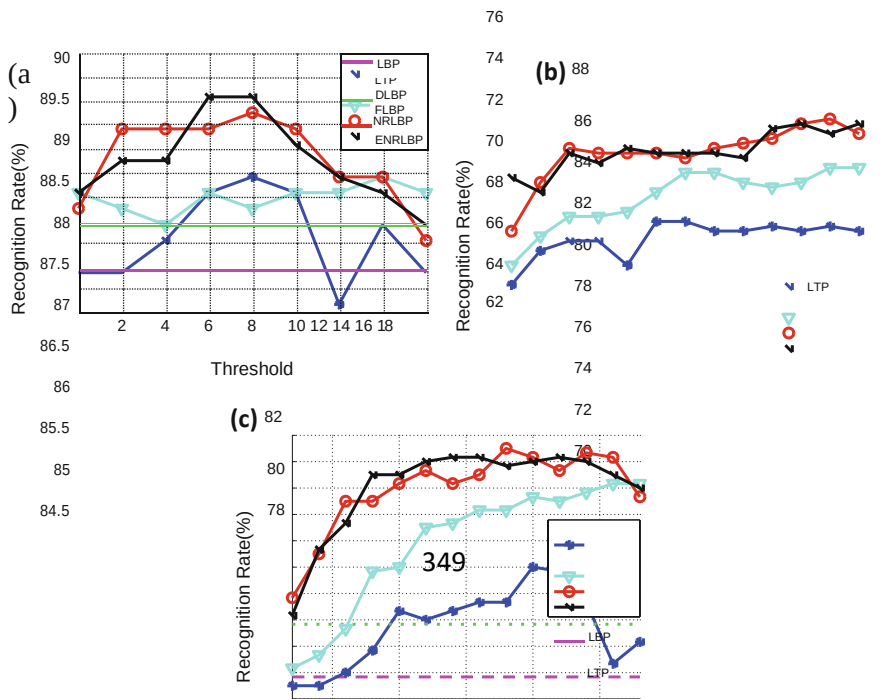
NRLBP	88.67	<u>86.33</u>	<u>81.00</u>	<u>89.33</u>	<u>85.67</u>	<u>79.67</u>	88.67	86.33	80.67
ENRLBP	<u>89.00</u>	86.00	80.33	88.00	85.67	78.67	<u>89.33</u>	<u>86.67</u>	<u>82.67</u>

$\sigma = 0.05, 0.10, 0.15$. The samples of noisy images are shown in Fig. 1.8. When the noise level is high, the images are barely recognizable and the recognition task becomes more challenging.

For LTP, NRLBP, and ENRLBP there is one free parameter: threshold $t \in [0, 255]$.

Fuzzy LBP also has a free parameter: fuzzification d . We vary t for LTP, NRLBP, and ENRLBP and d for fuzzy LBP. Only the recognition rates at the optimal setting are reported. The experimental results are summarized in Table 1.2. It can be seen that for most settings, NRLBP and ENRLBP outperform other approaches. When the noise level increases, the performance gain becomes more significant. This clearly demonstrates that NRLBP and ENRLBP are more resistant to noise compared with other LBP variants. We also observe that the experimental results for NRLBP and ENRLBP are fairly consistent for three different distance measures.

To study the effect of threshold t (or fuzzification parameter d), we plot the recognition rates versus t (or d) for LTP, FLBP, NRLBP, ENRLBP using Chi-square distance, as shown in Fig. 1.9. LBP and DLBP are shown as dashed lines. For low noise level, $\sigma = 0.05$, NRLBP, and ENRLBP are slightly better than DLBP and visibly better than LBP, LTP, and FLBP. For the middle and high noise levels, NRLBP and ENRLBP slightly outperform FLBP and significantly outperform LBP, LTP, and



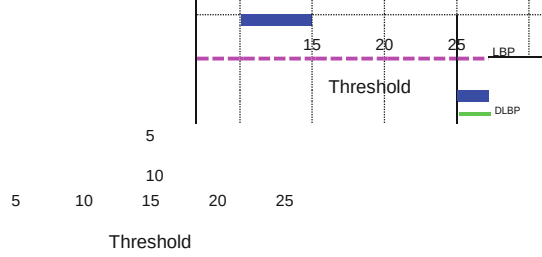


Fig. 1.9 The recognition rates of LBP, LTP, DLBP, FLBP, NRLBP, and ENRLBP using Chi-square distance versus threshold t on the AR database injected with Gaussian noise $\sigma = 0.05$ (a), $\sigma = 0.10$ (b), $\sigma = 0.15$ (c)

BLBP. Figure 1.9 shows that NRLBP, ENRLBP, and FLBP are the only ones that work well for all tested noise levels.

1.4.2 Face Recognition on the AR Database

For face recognition, we adopt a challenging experimental setting. Only one image per subject is used as the gallery (or training) set and all others are used as the probe set. In many real applications, we are not able to obtain multiple images per subject and we may have only one image per subject. 75 subjects are chosen from the AR database, each with 14 images. Each subject contains images from two sections. Each section contains 7 images: one neutral image, 3 images with different facial expressions, and 3 images in different illumination conditions. We repeat the experiments 6 times. For each trial, we use Image 1, 5, 6, 8, 12, 13 of each subject as the gallery set, respectively. The other 13 images of each subject are used as the probe set. It is a challenging experimental setting as face images with facial expression variations need to be identified based on a single face image.

Table 1.3 summarizes the average recognition rate and the standard derivation of each approach at the optimal setting on the AR database injected with Gaussian noise. Table 1.3 shows that NRLBP and ENRLBP achieve comparable or slightly better performance compared with FLBP, whereas they consistently outperform other approaches for all settings using different distance measures. As the noise level increases, the performance gain of NRLBP and ENRLBP over approaches other than FLBP becomes more significant.

1.4.3 Face Recognition on the Extended Yale B Database

The extended Yale B database [8, 18] contains 38 subjects under 9 poses and 64 illumination conditions. We follow the same database partition as in [40]. Images with the most neutral light source (“A+000E+00”) are used as gallery images and all other frontal images are used as probe images (in total 2414 images of 38 subjects). This dataset contains large illumination variations. The sample images are shown in the first row of Fig. 1.10. Some images are taken under extreme lighting conditions. Even after photometric normalization, as shown in the second row of Fig. 1.10, a large amount of image noise exists in the images. NRLBP and ENRLBP are compared with 6 LBP/LTP variants using nearest-neighbor classifier with Chi-square distance, histogram intersection, and modified G-statistic. Table 1.4 summarizes the highest recognition rates at the optimal threshold for various approaches using different distance measures. NRLBP and ENRLBP achieve slightly better performance than LBP, LTP, DLBP, and FLBP, and much better performance than NELBP and NTLBP.

Table 1.3 Summary of the average face recognition rate and the standard derivation of each approach at the optimal setting on the AR database injected with Gaussian noise

Algorithm	χ^2 distance, $\sigma =$			Histogram intersection, $\sigma =$			Modified G-statistic, $\sigma =$		
	0.05	0.10	0.15	0.05	0.10	0.15	0.05	0.10	0.15
LBP	83.44 % $\pm$ 1.44 %	64.65 % $\pm$ 2.92 %	40.91 % $\pm$ 6.52 %	81.64 % $\pm$ 1.58 %	55.18 % $\pm$ 7.37 %	34.34 % $\pm$ 4.38 %	79.04 % $\pm$ 1.71 %	56.63 % $\pm$ 3.02 %	34.56 % $\pm$ 5.93 %
LTP	83.91 % $\pm$ 1.03 %	65.09 % $\pm$ 5.88 %	43.78 % $\pm$ 9.96 %	81.85 % $\pm$ 2.08 %	55.26 % $\pm$ 12.07 %	37.69 % $\pm$ 10.77 %	80.26 % $\pm$ 0.90 %	62.58 % $\pm$ 2.44 %	42.74 % $\pm$ 3.74 %
DLBP	85.11 % $\pm$ 0.83 %	62.82 % $\pm$ 6.92 %	39.57 % $\pm$ 9.76 %	85.47 % $\pm$ 1.74 %	62.44 % $\pm$ 7.09 %	39.03 % $\pm$ 9.87 %	84.24 % $\pm$ 1.68 %	61.26 % $\pm$ 5.46 %	33.49 % $\pm$ 9.04 %
FLBP	83.95 % $\pm$ 1.32 %	78.19 % $\pm$ 1.08 %	71.04 % $\pm$ 1.70 %	84.17 % $\pm$ 1.42 %	74.26 % $\pm$ 2.01 %	59.86 % $\pm$ 3.06 %	81.44 % $\pm$ 1.41 %	74.97 % $\pm$ 1.45 %	68.77 % $\pm$ 2.14 %
NELBP	65.50 % $\pm$ 3.19 %	34.82 % $\pm$ 3.01 %	18.32 % $\pm$ 1.35 %	64.46 % $\pm$ 3.84 %	31.56 % $\pm$ 1.81 %	16.32 % $\pm$ 1.07 %	66.51 % $\pm$ 3.27 %	34.85 % $\pm$ 2.23 %	17.85 % $\pm$ 1.69 %
NTLBP	63.71 % $\pm$ 3.32 %	28.94 % $\pm$ 1.53 %	13.42 % $\pm$ 1.48 %	67.25 % $\pm$ 3.37 %	35.69 % $\pm$ 2.85 %	17.83 % $\pm$ 1.44 %	61.01 % $\pm$ 3.40 %	25.35 % $\pm$ 1.62 %	11.47 % $\pm$ 1.64 %
NRLBP	85.33 % $\pm$ 1.43 %	79.93 % $\pm$ 0.79 %	70.67 % $\pm$ 2.90 %	85.88 % $\pm$ 0.96 %	78.65 % $\pm$ 1.22 %	67.62 % $\pm$ 4.87 %	84.92 % $\pm$ 1.29 %	79.08 % $\pm$ 1.03 %	70.55 % $\pm$ 3.22 %

ENRLBP	<u>85.98 %</u> ± 1.35 %	<u>80.43 %</u> ± 0.97 %	<u>71.71 %</u> ± 1.79 %	<u>86.02 %</u> ± 1.09 %	<u>80.24 %</u> ± 1.55 %	<u>68.77 %</u> ± 3.05 %	<u>85.42 %</u> ± 1.03 %	<u>80.58 %</u> ± 0.72 %	<u>72.43 %</u> ± 1.50 %

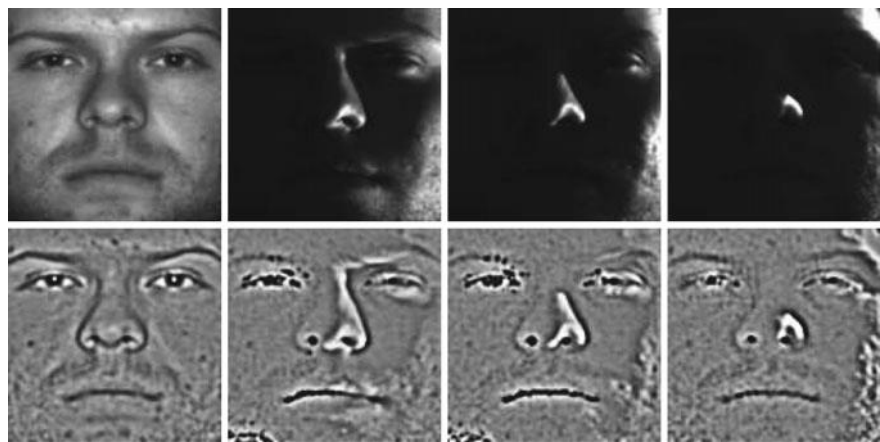


Fig. 1.10 The 1st row and 2nd row show the samples of geometrically normalized and photometrically normalized images for the extended Yale B database, respectively. The leftmost image is the gallery image, and the other 3 images taken under extreme lighting conditions are the probe images

Table 1.4 The face recognition rate and the optimal threshold on the extended Yale B database

Algorithm	Chi-square distance (%)	Histogram intersection (%)	Modified G-statistics (%)
LBP	96.07	93.32	96.12
LTP	98.25 (10)	97.99 (10)	98.29 (8)
DLBP	96.12	97.83	98.45
FLBP	98.45 (9)	98.16 (9)	98.54 (12)
NELBP	81.91	82.29	84.92
NTLBP	80.37	80.16	83.12
NRLBP	98.71 (9)	98.66 (8)	98.66 (11)
ENRLBP	98.75 (9)	98.66 (8)	98.62 (10)

1.4.4 Face Recognition on the O2FN Mobile Database

The O2FN mobile face database [33] is our in-house face database. It is designed to evaluate the face recognition algorithms on mobile face images, which are of low resolution and low image quality, and significantly corrupted by noise. It contains 2000 face images of size 144 176 pixels from 50 subjects. The images are self-taken by the users. The users are instructed to take about 20 indoor images and 20 outdoor images with minimum facial

expression variations and out-plane rotations. Thus, the O2FN database mainly contains in-plane rotations and illumination variations. Figure 1.11 shows some samples of geometrically normalized and photometrically normalized images. The images are captured by an O2 XDA frontal camera with native phone settings and without postprocessing. The images are severely distorted

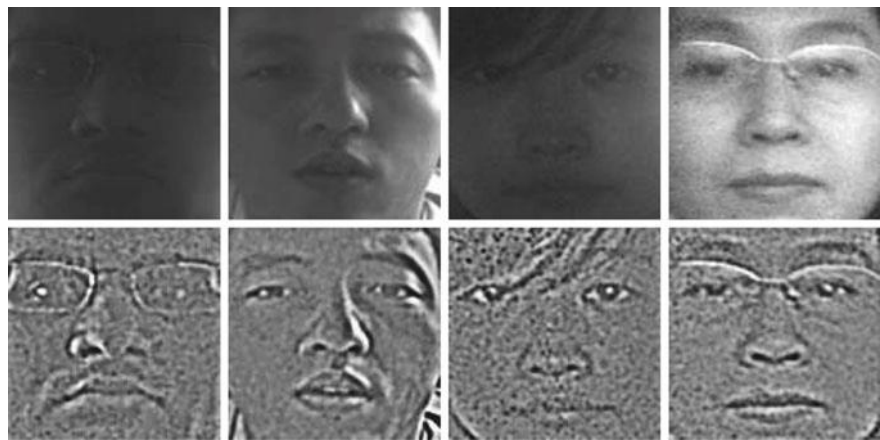


Fig. 1.11 The samples of geometrically normalized (*Row 1*) and photometrically normalized (*Row 2*) face images of the O2FN databases

Table 1.5 Performance comparison of face recognition on the O2FN database

Algorithm	Chi-square distance	Histogram intersection	Modified G-statistics
LBP	76.59 % $\pm$ 1.60 %	74.14 % $\pm$ 1.44 %	75.18 % $\pm$ 1.15 %
LTP	78.88 % $\pm$ 1.65 %	76.88 % $\pm$ 1.91 %	78.16 % $\pm$ 1.39 %
DLBP	78.07 % $\pm$ 1.69 %	79.88 % $\pm$ 2.10 %	79.01 % $\pm$ 2.04 %
FLBP	80.24 % $\pm$ 1.58 %	79.15 % $\pm$ 1.49 %	80.01 % $\pm$ 1.46 %
NELBP	56.74 % $\pm$ 1.75 %	58.25 % $\pm$ 2.16 %	59.12 % $\pm$ 1.73 %
NTLBP	56.96 % $\pm$ 1.82 %	58.40 % $\pm$ 1.65 %	58.63 % $\pm$ 2.18 %
NRLBP	80.76 % $\pm$ 1.56 %	80.29 % $\pm$ 1.63 %	80.68 % $\pm$ 1.57 %
ENRLBP	81.66 % $\pm$ 1.83 %	81.28 % $\pm$ 1.80 %	81.44 % $\pm$ 1.91 %

by noise, e.g., Gaussian noise, salt-and-pepper noise, and motion blur. To reduce the noise and illumination variations, the images are photometric normalized as in [40]. Even after photometric normalization, as shown in Fig. 1.11, the images still contain a large amount of noise.

NRLBP and ENRLBP are compared with 6 LBP/LTP variants using nearest-neighbor classifier with three different distance measures. The experiments are repeated five times. For each trial, we randomly choose one image of each subject as the gallery set and the rest as the probe set. We test LTP, NRLBP, and ENRLBP for different thresholds, and FLBP for

different d . Only the performance at the optimal setting is reported. The average recognition rates and the standard derivation at the optimal setting on the O2FN database are summarized in Table 1.5. NRLBP and ENRLBP achieve a comparable or slightly better performance compared with LTP, DLBP, and FLBP, and significantly outperform LBP, NELBP, and NTLBP using all three distance measures.

Table 1.6 The time consumption for feature extraction on the AR database

Algorithm	Time (ms)
LBP	10.8
LTP	16.4
DLBP	21.5
FLBP	161.7
NELBP	21.2
NTLBP	21.2
NRLBP	17.0
ENRLBP	23.1

1.4.5 Comparison of Computational Complexity

NRLBP and ENRLBP can be implemented by a lookup table to compute the NRLBP/ENRLBP histogram from uncertain code. It is very fast to compute the contribution of an *uncertain* code to the histogram by the lookup table and hence derive the feature vector of NRLBP/ENRLBP during recognition. The average time per image of feature extraction on the AR database for various LBP/LTP variants is shown in Table 1.6. The image is of size 128 128 pixels. Features are extracted under the setting of P 8, R 1. We use Matlab 2012b on Intel Duo CPU @3.0 GHz with 4 Gb RAM. Compared with LBP, NRLBP and ENRLBP introduce only small overhead. By contrast, it takes much more time to compute FLBP features, e.g., 161.7 ms, which is about 9.5 times that of NRLBP.

1.5 Conclusion

Facial expression and face recognition are two important functionalities of the virtual human and the social robot in order to achieve autonomous

behavior. LBP has been widely used for face and facial expression analysis; however, it is sensitive to noise. Even a small noise may change the LBP code significantly. Both LBP and its variants lack the mechanism to recover corrupted image local structures. In NRLBP, we assign the values of *uncertain* bits so as to form all possible uniform LBP codes. In this way we correct noisy nonuniform patterns back to uniform code. For LBP and LTP, a large group of local primitives, i.e., line patterns, is completely ignored. Thus, we proposed extended uniform patterns and formed them as our ENRLBP patterns when determining *uncertain* bits. NRLBP and ENRLBP show stronger noise-resistance than other approaches for face and facial expression recognition on various datasets.

References

1. Ahonen T, Hadid A, Pietikäinen M (2006) Face description with local binary patterns: application to face recognition. *IEEE Trans Pattern Anal Mach Intell* 28(12):2037–2041
2. Ahonen T, Pietikäinen M (2007) Soft histograms for local binary patterns. In: *Proceedings of the finnish signal processing symposium (FINSIG 2007)*, pp 1–4
3. Akhloufi MA, Bendada A (2010) Locally adaptive texture features for multispectral face recognition. In: *IEEE international conference on systems man and cybernetics (SMC)*, IEEE, pp 3308–3314
4. Boland MV, Murphy RF (2001) A neural network classifier capable of recognizing the patterns of all major subcellular structures in fluorescence microscope images of hela cells. *Bioinformatics* 17(12):1213–1223
5. Fathi A, Nilchi ARN (2012) Noise tolerant local binary pattern operator for efficient texture analysis. *Pattern Recogn Lett* 33:1093–1100
6. Froba B, Ernst A (2004) Face detection with the modified census transform. In: *Proceedings of sixth IEEE international conference on automatic face and gesture recognition*, IEEE, pp 91–96
7. Fu X, Wei W (2008) Centralized binary patterns embedded with image euclidean distance for facial expression recognition. In: *International conference on natural computation*, vol 4, IEEE, pp 115–119
8. Georgiades AS, Belhumeur PN, Kriegman DJ (2001) From few to many: illumination cone models for face recognition under variable lighting and pose. *IEEE Trans Pattern Anal Mach Intell* 23(6):643–660
9. Guo Z, Zhang L, Zhang D (2010) A completed modeling of local binary pattern operator for texture classification. *IEEE Trans Image Process* 19(6):1657–1663
10. Gupta R, Patil H, Mittal A (2010) Robust order-based methods for feature description. In: *IEEE conference on computer vision and pattern recognition (CVPR)*, pp 334–341
11. Hafiane A, Seetharaman G, Zavidovique B (2007) Median binary pattern for textures classification. In: *Image analysis and recognition*, Springer, pp 387–398
12. He W, Ahonen T, Pietikäinen M (2008) A bayesian local binary pattern texture descriptor. In: *International conference on pattern recognition*, IEEE, pp 1–4
13. He D, Cercone N (2009) Local triplet pattern for content-based image retrieval. In: *Image analysis and recognition*, pp 229–238
14. Heikkilä M, Pietikäinen M, Schmid C (2009) Description of interest regions with local binary patterns. *Pattern Recogn* 42(3):425–436
15. Huang X, Li SZ, Wang Y (2004) Shape localization based on statistical method using extended local binary pattern. In: *IEEE first symposium on multi-agent security and survivability*, IEEE, pp 184–187
16. Iakovidis DK, Keramidas EG, Maroulis D (2008) Fuzzy local binary patterns for ultrasound texture characterization. In: *Image analysis and recognition*
17. Keramidas E, Iakovidis D, Maroulis D (2011) Fuzzy binary patterns for uncertainty-aware texture representation. *Electron Lett Comput Vis Image Anal* 10(1):63–78
18. Lee KC, Ho J, Kriegman DJ (2005) Acquiring linear subspaces for face recognition under variable lighting. *IEEE Trans Pattern Anal Mach Intell* 27(5):684–698
19. Li X, Weiming H, Zhang Z, Wang H (2010) Heat kernel based local binary pattern for face representation. *IEEE Signal Process Lett* 17(3):308–311
20. Li Z, Liu G, Yang Y, You J (2012) Scale-and rotation-invariant local binary pattern using scale-adaptive texton and subuniform-based circular shift. *IEEE Trans Image Process* 21(4):2130–2140

21. Liao W-H, Young T-J (2010) Texture classification using uniform extended local ternary patterns. In: 2010 IEEE international symposium on multimedia (ISM), IEEE, pp 191–195
22. Liao S, Zhao G, Kellokumpu V, Pietikäinen M, Li SZ (2010) Modeling pixel process with scale invariant local patterns for background subtraction in complex scenes. In: IEEE conference on computer vision and pattern recognition (CVPR), IEEE, pp 1301–1306

23. Li B, Meng MQ-H (2009) Texture analysis for ulcer detection in capsule endoscopy images. *Image Vis comput* 27(9):1336–1342
24. Martinez AM, Benavente R (1998) The AR face database. CVC Tech. Report #24
25. Max LS, Law WK, Chung ACS (2009) Dominant local binary patterns for texture classification. *IEEE Trans Image Process* 18(5):1107–1118
26. Mu Y, Yan S, Liu Y, Huang T, Zhou B (2008) Discriminative local binary patterns for human detection in personal album. In: *Proceeding of IEEE conference on computer vision and pattern recognition*, pp 1–8
27. Nanni L, Lumini A, Brahnam S (2010) Local binary patterns variants as texture descriptors for medical image analysis. *Artif Intell Med* 49(2):117–125
28. Nanni L, Brahnam S, Lumini A (2012) A simple method for improving local binary patterns by considering non-uniform patterns. *Pattern Recogn* 45(10):3844–3852
29. Ojala T, Pietikäinen M, Harwood D (1996) A comparative study of texture measures with classification based on featured distributions. *Pattern Recogn* 29(1):51–59
30. Ojala T, Maenpää T, Pietikainen M, Viertola J, Kyllonen J, Huovinen S (2002) Outex-new framework for empirical evaluation of texture analysis algorithms. *Proc Int Conf Pattern Recogn* 1:701–706
31. Ojala T, Pietikäinen M, Maenpää T (2002) Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans Pattern Anal Mach Intell* 24(7):971–987
32. Pietikäinen M, Hadid A, Zhao G, Ahonen T (2011) *Computer vision using local binary patterns*, vol 40. Springer, Berlin
33. Ren J, Jiang X, Yuan J (2013) A complete and fully automated face verification system on mobile devices. *Pattern Recogn* 46(1):45–56
34. Ren J, Jiang X, Yuan J (2013) Noise-resistant local binary pattern with an embedded error-correction mechanism. *IEEE Trans Image Process* 22(10):4049–4060
35. Ren J, Jiang X, Yuan J (2013) Dynamic texture recognition using enhanced lbp features. In: *IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp 2400–2404
36. Ren J, Jiang X, Yuan J (2013) Learning binarized pixel-difference pattern for scene recognition. In: *IEEE international conference on image processing (ICIP)*, pp 2494–2498
37. Ren J, Jiang X, Yuan J (2013) Relaxed local ternary pattern for face recognition. In: *IEEE international conference on image processing (ICIP)*, pp 3680–3684
38. Satpathy A, Jiang X, Eng H-L (2014) LBP based edge-texture features for object recognition. *IEEE Trans Image Process* 23(5):1953–1964
39. Tan N, Huang L, Liu C (2009) A new probabilistic local binary pattern for face verification. In: *Proceeding of IEEE international conference on image processing*, pp 1237–1240
40. Tan X, Triggs B (2010) Enhanced local texture feature sets for face recognition under difficult lighting conditions. *IEEE Trans Image Process* 19(6):1635–1650
41. Trefný J, Matas J (2010) Extended set of local binary patterns for rapid object detection. In: *Proceedings of the computer vision winter workshop*, vol 2010
42. Wang X, Han TX, Yan S (2009) An HOG-LBP human detector with partial occlusion handling. In: *Proceeding of IEEE international conference on computer vision*, pp 32–39
43. Wu J, Rehg JM (2011) CENTRIST: a visual descriptor for scene categorization. *IEEE Trans Pattern Anal Mach Intell* 33(8):1489–1501
44. Xiao Y, Jianxin W, Yuan J (2014) mCENTRIST: a multi-channel feature generation mechanism for scene categorization. *IEEE Trans Image Process* 23(2):823–836
45. Yao C-H, Chen S-Y (2003) Retrieval of translated, rotated and scaled color textures. *Pattern Recogn* 36(4):913–929
46. Zhang W, Shan S, Gao W, Chen X, Zhang H (2005) Local gabor binary pattern histogram

- sequence (lgbphs): a novel non-statistical model for face representation and recognition. In: IEEE international conference on computer vision, vol 1, IEEE, pp 786–791
47. Zhao G, Ahonen T, Matas J, Pietikäinen M (2012) Rotation-invariant image and video description with local binary pattern features. *IEEE Trans Image Process* 21(4):1465–1477

48. Zhao S, Gao Y, Zhang B (2008) Sobel-LBP. In: IEEE international conference on image processing, IEEE, pp 2144–2147
49. Zhao G, Pietikäinen M (2007) Dynamic texture recognition using local binary patterns with an application to facial expressions. *IEEE Trans Pattern Anal Mach Intell* 29(6):915–928
50. Zhou H, Wang R, Wang C (2008) A novel extended local-binary-pattern operator for texture analysis. *Inf Sci* 178(22):4314–4325

Chapter 2

Body Movement Analysis and Recognition

Yang Xiao, Hui Liang, Junsong Yuan and Daniel Thalmann

Abstract In this chapter, a nonverbal way of communication for human–robot inter- action by understanding human upper body gestures will be addressed. The human– robot interaction system based on a novel combination of sensors is proposed. It allows one person to interact with a humanoid social robot with natural body lan- guage. The robot can understand the meaning of human upper body gestures and express itself by using a combination of body movements, facial expressions, and verbal language. A set of 12 upper body gestures is involved for communication. Human–object interactions are also included in these gestures. The gestures can be characterized by the head, arm, and hand posture information. CyberGlove II is employed to capture the hand posture. This feature is combined with the head and arm posture information captured from Microsoft Kinect. This is a new sensor solution for human-gesture capture. Based on the body posture data, an effective and real- time human gesture recognition method is proposed. For experiments, a human body gesture dataset was built. The experimental results demonstrate the effectiveness and efficiency of the proposed approach.

2.1 Introduction

Recently, human–robot interaction (HRI) has drawn the attention of the academic and industrial communities. Regarded as the sister community of human–computer interaction (HCI), HRI is still a relatively young field that began to emerge in the

Y. Xiao (**B**)

School of Automation, Huazhong University of Science and Technology, Wuhan, China

e-mail: Yang_Xiao@hust.edu.cn

H. Liang J. Yuan D. Thalmann

BeingThere Centre, Nanyang Technological University, Singapore, Singapore

J. Yuan

e-mail: JSYUAN@ntu.edu.sg

D. Thalmann

e-mail: danielthalmann@ntu.edu.sg

© Springer International Publishing Switzerland 2016

31

N. Magnenat-Thalmann et al. (eds.), *Context Aware Human-Robot and Human-Agent Interaction*, Human–Computer Interaction Series, DOI 10.1007/978-3-319-19947-4_2



Fig. 2.1 Human–robot social interaction, with human on the *right* and robot on the *left*

1990s [10, 14]. It is an interdisciplinary research field that requires contributions from mathematics, psychology, mechanical engineering, biology, computer science, etc. [14].

HRI aims to understand and shape the interactions between humans and robots. Unlike early interactions, more social dimensions must be considered in HRI, especially when interactive social robots are involved [10, 13]. In this case, robots should be believable. Moreover, humans prefer to interact with robots as they do with other people [10, 13]. Therefore, one way to increase believability would be to make the robot interact with humans using the same modalities as human–human interaction. This includes verbal and body language as well as facial expressions; i.e., the robots should be able to use these modalities for both perception and expression. Some social robots have already been proposed toward this goal. For instance, the Leonardo robot expresses itself using a combination of voice, facial, and body expressions [25]. Another example is the Nao humanoid robot¹ that can use vision along with gestures and body expression of emotions [2]. Different from these two robots, the Nadine robot is a highly realistic humanoid robot (Fig. 2.1). This robot presents some different social challenges. In this chapter, a human–robot interaction system that addresses some of these challenges is proposed. As shown in Fig. 2.1, it supports a person to communicate and interact with a humanoid robot. In the proposed system, the human can naturally communicate

with the Nadine robot using body language. The Nadine robot is able to express herself by using a combination of speech, body language, and facial expressions. In this chapter, the main research concern addressed is how to establish communication between human and robot using body language.

Verbal and nonverbal language are two means of communication for human– human interaction. Verbal language has been used in many HRI systems

¹<http://www.aldebaran-robotics.com/>

Fig. 2.2 Human–human interaction accompanied with nonverbal language



[11, 21, 23, 26–28]; however, it still has some constraints. That is, speech recognition accuracy is likely to be affected by the background noise, human accents, and device performance. Moreover, learning and interpreting the subtle rules of syntax and grammar in speech is a difficult task. These factors limit the practical use of verbal language to a certain degree. On the other hand, nonverbal clues can convey rich communication messages [7, 19]. Evidently, they play an important role in

human–human interaction to reinforce the communication performance as shown in Fig. 2.2. Thus, one of our research motivations is *to apply nonverbal language to human–robot social interaction*. More specifically, upper body gesture language is employed. Currently, 12 human upper body gestures are involved in the proposed system, which are all natural gestures with intuitive semantics. They are characterized by head, arm, and hand posture information simultaneously. It is worth noting that *human–object* interactions are involved in these gestures. *Human–object* interaction events manifest frequently during human–human interaction in daily life. However, to our knowledge, they were largely ignored by previous HRI systems.

The main challenge to apply upper body gesture language to human–robot interaction is how to enable the Nadine robot to understand and react to human gestures accurately and in real-time. To achieve this goal, two crucial issues need to be solved:

First, an appropriate human gesture-capture sensor solution is required. To recognize the 12 upper body gestures, head, arm, and hand posture information is needed simultaneously. As robustly obtaining hand posture based on vision-based sensors (such as the RGB camera) is still a difficult task [18, 29], the wearable Cyber-Glove II [15] (shown in Fig. 2.3) is used. Using this device, high-accuracy hand

Fig. 2.3 CyberGlove II**Fig. 2.4** The microsoft kinect RGB and depth sensor

posture data can be acquired stably. Meanwhile, Microsoft Kinect [24] (shown in Fig. 2.4) is an effective and efficient low-cost depth sensor applied successfully to human body tracking. The skeletal joints can be extracted from the Kinect depth images [24] in real-time (30 fps). In our work, Kinect is applied to capture the upper body (head and arm) posture information. Recently, Kinect 2 that supports tracking multiple people with better depth imaging quality was released. Since our work investigates the HRI scenario involving only one person, Kinect is sufficient to handle the human body tracking task;

- Second, an effective and real-time gesture recognition method should be developed. Based on the CyberGlove II and Kinect posture data, descriptive upper body gesture feature is proposed. To leverage the gesture understanding performance, LMNN distance metric learning method [33] is applied. Then, the energy-based LMNN classifier is used to recognize the gestures.

To evaluate the proposed gesture recognition method, a human upper body gesture dataset is constructed. This dataset contains gesture samples from 25 people of different genders, body sizes, and culture

backgrounds. The experimental results demonstrate the effectiveness and efficiency of our method.

Overall, the main contributions of this chapter include:

- A novel human gesture-capture sensor solution is proposed. That is, the Cyber- Glove II and Kinect are integrated to capture head, arm, and hand posture information simultaneously;
- An effective and real-time upper body gesture recognition approach is proposed; To support humans to communicate and interact with robots using body language, a gesture understanding and human–robot interaction (GUHRI) system is built.

The remainder of this chapter is organized as follows. Problematic issues are discussed in Sect. 2.2. Section 2.3 gives an overview of the related state-of-the-art works. The recent approaches are described in Sect. 2.4. Section 2.5 introduces the future avenues. The conclusions are drawn in Sect. 2.6.

2.2 Problematic

To successfully apply nonverbal language to HRI by understanding human upper body gestures, some critical problematic issues and challenges need to be addressed. A brief discussion of this point has been made in Sect. 2.1. In this section, we extend the discussion from the perspectives of the HRI system in detail.

As our research aims to promote social interaction between humans and robot, the semantics of natural human body language need to be analyzed. To fully understand the human upper body gestures, head, arm, and hand posture information should be captured simultaneously in an effective way. This proposition is effectively demonstrated by Figs. 2.1 and 2.2. However, to our knowledge, very few previous body gesture recognition works take hand and rough body posture information into consideration together. Thus, appropriate human gesture-capture devices are the essential components to construct a successful HRI system. For human gesture capture, vision-based sensors are the trend as they exert no burden on the users and lead to better user experience. Some successful examples have already emerged recently, such as Microsoft Kinect, which is applied to human body parsing and tracking. However, under unconstrained conditions, it is still difficult to capture the hand posture robustly because of drastic hand rotation and serious occlusion as shown in Fig. 2.5. Meanwhile, it is not feasible to restrict the user's hand position and orientation during the phase of natural HRI. As a consequence, according to the current capacity of vision-based sensors, they are not the optimal choice to capture the hand posture for the HRI system. Thus, a more applicable human gesture-capture sensor solution should be proposed. In addition to effectiveness,

Fig. 2.5 The different hand gestures with drastic rotation and serious occlusion



another important factor for human gesture capture is efficiency. As HRI is in high real-time demand, the data acquisition stage must be finished as soon as possible. Based on the reliable hand and body posture feature, how to recognize the upper body gestures effectively in real-time is the latest concern that needs to be addressed. Since the hand and rough body posture information is captured from different sensors, i.e., they are multimodal data, how to fuse them to form a unified body gesture description is the first point we focus on. Second, for HRI application an adequate classification scheme should be proposed to leverage the performance, including the distance metric learning method and the choice of classifier. Last but not least, enough training samples are required to drive the supervised body gesture recognition approach. However, there is no existing body gesture dataset that can be applied to our work directly. Thus, building a novel upper body gesture dataset with sufficient available samples is another crucial task.

Whether robots can naturally react to human body language will largely affect the user experience. Since the Nadine robot has the capacity to express herself using a combination of speech, body language, and facial expressions, a suitable interaction scenario is required for robot control to make her more humanlike and vivid. Indeed, the interaction scenario should be designed according to the human habits during human–human interaction.

2.3 State of the Art

HRI systems are constructed mainly based on verbal, nonverbal, or multimodal communication modalities. As mentioned in Sect. 2.1, verbal language still faces constraints in practical applications. Our work focuses on studying how to apply nonverbal language to human–robot social interaction, especially using upper body gesture language. Some HRI systems have already employed body gesture language for human–robot communication. In [30], an arm gesture-based interface for HRI was proposed. The user could control a mobile robot using static or dynamic arm gestures. Hand gesture was used as the communication modality for

HRI in [5]. The HRI systems addressed in [27, 28] could recognize the human pointing gesture using the 3D head and hand position information, and head orientation was further appended to leverage the performance. In [11], the social robot could understand the human body language characterized by arm and head posture. Our proposition on nonverbal human–robot communication is different from previous works mainly in two aspects. First, head, arm, and hand posture are jointly captured to describe the 12 upper body gestures involved in the GUHRI system. Second, the gestures accompanied with human–object interaction can be understood by the robot. These gestures were always ignored by previous HRI systems, although they manifest frequently in daily life as shown in Fig. 2.6.



Fig. 2.6 Human–object interactions in daily life

Body gesture recognition plays an important role in the GUHRI system. According to the gesture-capture sensor type, gesture recognition systems can be categorized as encumbered and unencumbered [4]. Encumbered systems require the user to wear physical assistive devices such as infrared responders, hand markers, or data gloves. These systems have high precision and fast response, and are robust to environmental changes. Many encumbered systems have been proposed. For instance, two education systems [1] were built for the deaf using data gloves and optical motion capture devices; Lu et al. [18] proposed an immersive virtual object manipulation system based on two data gloves and a hybrid ultrasonic tracking system. Although most commercialized gesture-capture devices are currently encumbered, unencumbered systems are expected to be the future choice, especially vision-based systems. With the emergence of low-cost 3D vision sensors, the application of such devices becomes a hot topic in both the research and commercial fields. One of the most famous examples is Microsoft Kinect, which has been successfully employed in human body tracking [24], activity analysis [31], and gesture understanding [37, 38]. Even for other vision applications (such as scene categorization [36] and image segmentation [34, 35]), Kinect holds the potential to boost the

performance. However, accurate and robust hand posture capture is still a difficult task for vision-based sensors.

As discussed above, both encumbered and unencumbered sensors possess intrinsic advantages and drawbacks. For specific applications, they can be complementary. In the GUHRI system a tradeoff between the two kinds of sensors is made, that is, the fine hand posture is captured by the encumbered device (CyberGlove II), while the rough upper body posture is handled by the unencumbered sensor (Kinect).

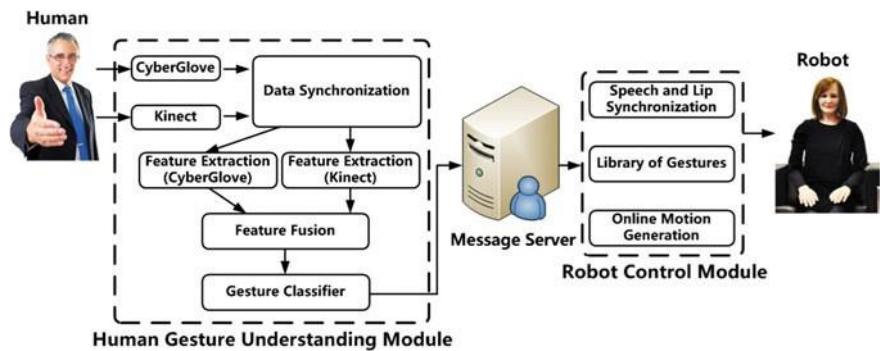


Fig. 2.7 The GURHI system architecture

2.4 Recent Approaches

In this section, we give an overview of the GURHI system. The human upper body gesture understanding method is then illustrated. Next, the scenario for HRI is intro- duced. Experiment and discussion are finally given.

2.4.1 System Overview

The proposed GURHI system is able to capture and understand human upper body gestures and trigger the robot’s reaction in real-time accordingly. The GURHI sys- tem is implemented using a framework called Integrated Integration Platform (I2P) specifically developed for integration. I2P was developed by the Institute for Media Innovation.² This framework allows for the link and integration of perception, deci- sion, and action modules within a unified and modular framework. The platform uses client–server communications between the different components. Each com- ponent has an I2P interface and communication between the client and servers is implemented using thrift.³ It should be noted that the framework is highly modular and components can be added to make the GURHI system extendable. As shown in Fig. 2.7, the current GURHI system is mainly composed of two modules. One, the *human gesture*

understanding module that serves as the communication interface between human and robot, and the other is the *robot control module* proposed to control the robot's behaviors for interaction. At this stage, our system supports the interaction between one person and one robot.

One right-handed CyberGlove II and one Microsoft Kinect are employed to capture the human hand and body posture information simultaneously for gesture

²<http://imi.ntu.edu.sg/Pages/Home.aspx>

³<http://thrift.apache.org/>

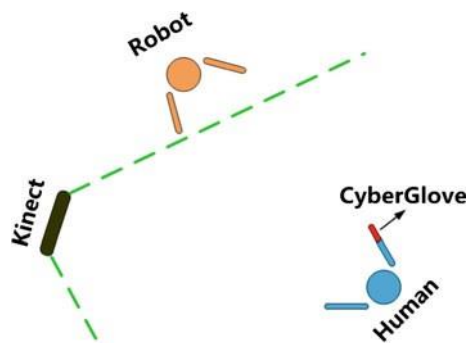
understanding. This is a new gesture capturing sensor solution, different from all the approaches introduced in Sect. 2.3. Specifically, CyberGlove is used to capture the hand posture, and Kinect is applied to acquire the 3D position information of the human skeletal joints (including head, shoulder, limb, and hand). At this stage, the GUHRI system relies on the upper body gestures triggered by the human right hand and right arm.

Apart from the CyberGlove, the user does not need to wear any other device. Thus, the proposed sensor solution does not exert a heavy burden to make the user uncomfortable. Meanwhile, as the CyberGlove II is involved in the system using Bluetooth, the user can move freely. In addition, the GUHRI system is able to recognize gestures with human–object interaction, such as “call”, “drink”, “read” and “write” by fusing the hand and body posture information. These gestures were often ignored by previous systems. However, they manifest frequently during the daily interaction between humans. These affect the interaction state abruptly and should be considered as essential elements of the natural HRI. In our system, the robot is able to recognize and give meaningful responses to these gestures (Fig. 2.8).

The first step of the gesture understanding phase is to synchronize the original data from CyberGlove and Kinect. The descriptive features are then extracted from them respectively. The multimodal features are then fused to generate the unified input for the gesture classifier. Lastly, the gesture recognition and understanding results are sent to the robot control module via message server to trigger the robot’s reaction.

The robot control module enables the robot to respond to the human’s body gesture language. In our system, the robot’s behavior is composed of three parts: body movement, facial expression, and verbal language. Combining these modalities makes the robot more lifelike, and should enhance the users’ interest during interaction.

Fig. 2.8 The GUHRI system deployment



2.4.2 Human Upper Body Gesture Understanding

As an essential part of the GUHRI system, the human upper body gesture understanding module plays an important role during interaction. Its performance will highly affect the interaction experience. In this section, our upper body gesture understanding method by fusing the gesture information from CyberGlove and Kinect is illustrated in detail. First, the body gestures included in the GUHRI system are introduced. The feature extraction pipelines for both CyberGlove and Kinect are then presented. To generate an integral gesture description, the multimodal features from different sensors are fused as the input for classifier. Aiming to enhance the gesture recognition accuracy, LMNN distance metric learning approach [33] is applied for mining the optimal distance measures, and the energy-based classifier [33] is applied for decision making.

2.4.2.1 Gestures in the GUHRI System

At the current stage, 12 static upper body gestures are included in the GUHRI system. As we only have one right-hand CyberGlove, to obtain accurate hand posture information all the gestures are mainly triggered by the human right hand and right arm. The involved gestures can be partitioned into two categories, according to whether human–object interaction happens:

- *Category 1*: body gestures *without* human–object interaction;
- *Category 2*: body gestures *with* human–object interaction.

Category 1 contains 8 upper body gestures: “*be confident*”, “*have question*”, “*object*”, “*praise*”, “*stop*”, “*succeed*”, “*shake hand*” and “*weakly agree*.” Some gesture samples are shown in Fig. 2.9. These gestures are natural and have intuitive meaning. They are related to the human’s emotional state and behavior intention and are not ad hoc for specific applications. Therefore, gesture-to-meaning mapping is not needed in our system. As human behavior habits are not all the same, recognizing natural gestures is more challenging than ad hoc ones. However, natural gestures are more meaningful for HRI. As shown in Fig. 2.9, both hand and body

posture information are required for recognizing these gestures. For instance, the upper body postures corresponding to “*have question*” and “*object*” are very similar. Without the hand posture, they are difficult to distinguish. The same happens to “*have question*,” “*weakly agree*,” and “*stop*.” That is, they correspond to similar hand gestures but very different upper body postures.

Category 2 is composed of four other upper body gestures: “*call*,” “*drink*,” “*read*,” and “*write*” (Fig. 2.10). Being different from *Category 1* gestures, these four gestures happen with human–object interactions. Existing systems do not consider such gestures (see Sect. 2.3). One main reason is that objects often cause body occlusion, especially to the hand. In this case, vision-based hand gesture recognition methods are impaired. Hence, the CyberGlove is employed to capture the hand posture. In the



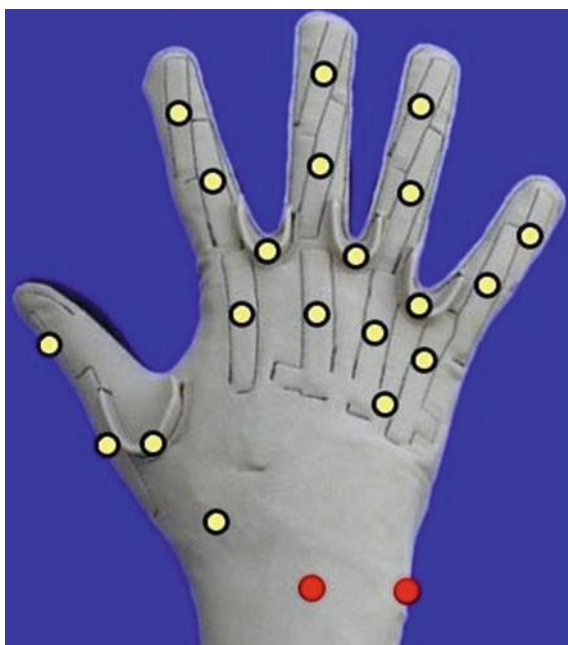
Fig. 2.9 The *Category 1 upper body* gestures. These gestures can be characterized by the body and hand posture information simultaneously



Fig. 2.10 The *Category 2 upper body* gestures. These gestures can be characterized by the body and hand posture information simultaneously

GUHRI system, *Category 2* gestures are recognized and affect the interaction in a realistic way. These gestures are also recognized based on the hand and upper body posture information.

Fig. 2.11 CyberGlove II data joints [15]



2.4.2.2 Feature Extraction and Fusion

In this section, we introduce the feature extraction methods for both human hand and upper body posture description. The multimodal feature fusion approach is also illustrated.

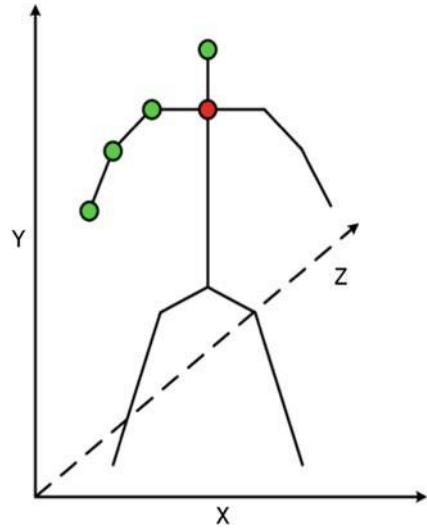
(a) Hand Posture Feature

As discussed above, the description of the human hand and upper body posture is key to recognize and understand the 12 upper body gestures.

The immersion wireless CyberGlove II is used as the hand posture capture device in the GUHRI system. As one of the most sophisticated and accurate data gloves, CyberGlove II provides 22 high-accuracy joint-angle measurements in real-time. These measurements reflect the bending degree of the fingers and wrist. The 22 data joints (marked as big white or black dots) are located on the CyberGlove as shown in Fig. 2.11. However, not all the joints are used. For the hand gestures in our application, we found that the wrist posture does not provide stable descriptive information. The wrist bending degrees of different people vary to a large extent even for the same gesture. This phenomenon is related to different behavior habits. This is the reason that the two wrist data joints (marked as black) are discarded. A 20-dimensional feature vector F_{hand} is extracted from the 20 white data joints to describe the human hand posture as

$$F_{\text{hand}} = (h_1, h_2, h_3 \dots h_{19}, h_{20}), \quad (2.1)$$

Fig. 2.12 The selected body skeletal joints



where h_i is the bending degree corresponding to the white data joint i .

(b) Upper Body Posture Feature

Using the Kinect sensor, we shape the human upper body posture intermediately using the 3D skeletal joint positions. For a full human subject, 20 body joint positions can be detected and tracked by the real-time skeleton tracker [24] based on the Kinect depth frame. This is invariant to posture, body shape, clothing, etc. Each joint J_i is represented by three coordinates at the frame t as

$$J_i = (x_i(t), y_i(t), z_i(t)). \quad (2.2)$$

However, not all the 20 joints are necessary for upper body gesture recognition. As aforementioned, head and right arm are highly correlated with the 12 upper body gestures (Figs. 2.9 and 2.10). For efficiency, only four upper body joints are chosen as the descriptive joints for gesture understanding. These are “*head*,” “*right shoulder*,” “*right elbow*,” and “*right hand*” that are shown as gray dots in Fig. 2.12.

Directly using the original 3D joint information for body posture description is not stable, because it is sensitive to the relative position between human and Kinect. Solving this problem by restricting the human’s position is not appropriate for interaction. In [31], human action is recognized by using the pairwise relative positions between all joints, which is robust to the human–Kinect relative position. Inspired by this work, a simplified solution is proposed. First, the “*middle of the two shoulders*” joint (black dot in Fig. 2.12) is selected as the reference joint. The pairwise relative positions between the four descriptive joints and the reference joint are then computed for body posture description as

$$J_{sr} = J_s - J_r, \quad (2.3)$$

where J_s is the descriptive joint and J_r is the reference joint. With this processing, J_{sr} is less sensitive to the human–Kinect relative position. It is mainly determined by the body posture. The “*middle of the two shoulders*” was chosen as the reference joint because it can be robustly detected and tracked in most cases. Moreover, it is rarely occluded by the limbs or the objects when the gestures in GUHRI system happen. Finally, an upper body posture feature vector F_{body} of 12 dimensions is constructed by combining the four pairwise relative positions as

$$F_{\text{body}} = (J_{1r}, J_{2r}, J_{3r}, J_{4r}), \quad (2.4)$$

where J_{1r}, J_{2r}, J_{3r} and J_{4r} are the pairwise relative positions.

(c) Feature Fusion

From CyberGlove II and Kinect, two multimodal feature vectors: F_{hand} and F_{body} are extracted to describe the hand posture and upper body posture respectively. To fully understand the upper body gestures, the joint information about the two feature vectors is required. Both are essential for the recognition task. However, the two feature vectors locate in different value ranges. Simply combining them as the input for classifier will yield performance bias on the feature vector of low values. To overcome this difficulty, we scale them into similar ranges before feature fusion. Suppose F_i is one dimension of F_{hand} or F_{body} , $F_i^{\max}$ and $F_i^{\min}$ are the corresponding

$$i \quad i$$

maximum and minimum value in the training set. Then F_i can be normalized as

$$\hat{F}_i = \frac{F_i - F_i^{\min}}{F_i^{\max} - F_i^{\min}}, \quad (2.5)$$

for both training and test.

After normalization, the effectiveness of the two feature vectors for gesture recognition will be balanced. Finally, they are fused to generate an integral feature vector by concatenation as

$$\mathbf{F} = (\hat{F}_{\text{hand}}, \hat{F}_{\text{body}}). \quad (2.6)$$

This process results in a 32-dimensional feature vector $\mathbf{F}$ used for upper body gesture recognition.

2.4.2.3 Classification Method

Using $\mathbf{F}$ as the input feature, the upper body gestures will be recognized by template matching based on the *energy-based LMNN classifier* proposed in [33].⁴ It is derived from the energy-based model [8] and the LMNN distance metric learning method [33]. The latter part is the key to constructing this classifier. LMNN distance metric learning approach is proposed to seek the best distance measure for the

⁴The source code is available at <http://www.cse.wustl.edu/~kilian/code/lmnn/lmnn.html>

k -nearest neighbor (KNN) classification rule [9]. As one of the oldest methods for pattern recognition, KNN classifier is simple to implement and use. Nevertheless, it can still yield comparative results in certain domains such as object recognition and shape matching [3], it has also been applied to action recognition [20].

The KNN rule classifies each testing sample by the majority label voting among its k -nearest training samples. Its performance crucially depends on how to compute the distances between different samples for the k nearest neighbors search. Euclidean distance is the most widely used distance measure, although it ignores any statistical regularities that may be estimated from the training set. Ideally, the distance measure should be adjusted according to the specific task being solved. To achieve better classification performance, LMNN distance metric learning method is proposed to mine the best distance measure for the KNN classification.

Let $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be a training set of n labeled samples with inputs $\mathbf{x}_i \in \mathbb{R}^d$ and class labels y_i .

The main goal of LMNN distance metric learning is to learn a linear transformation $\mathbf{L} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ that is used to compute the square sample distances as

$$D(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)\|_2^2 \quad (2.7)$$

Using $D(\mathbf{x}_i, \mathbf{x}_j)$ as the distance measure tends to optimize the KNN classification by making each input $\mathbf{x}_i$ have k nearest neighbors that share the same class label y_i to the greatest possibility. Figure 2.13 gives an intuitive illustration on LMNN distance metric learning. Compared with Euclidean distance, LMNN distance tries to pull the nearest neighbors of class y_i closer to $\mathbf{x}_i$, while pushing the neighbors from different classes away. On the assumption that the training set and the test set keep the similar feature distribution, LMNN distance metric learning can help to improve the KNN classification result.

The energy-based LMNN classifier makes use of both the $D(\mathbf{x}_i, \mathbf{x}_j)$ distance measure and the loss function defined for LMNN distance metric learning. It constructs an energy-based criterion function, and the testing sample is assigned to the class that yields the minimum loss value. As the related theory is sophisticated, we do not give a detailed definition of the energy-based LMNN classifier here; readers can turn to [33] for reference.

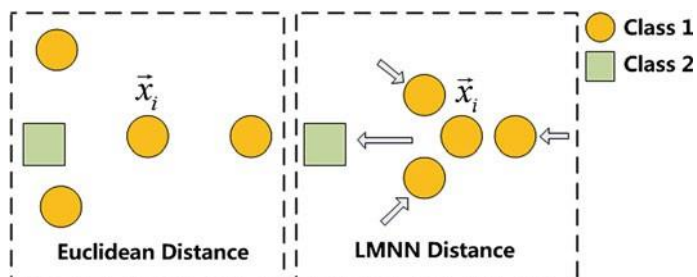


Fig. 2.13 Illustration of the LMNN distance metric learning



Fig. 2.14 Examples of body movements and facial expressions from the library of gestures

2.4.2.4 Human–Robot Interaction

As a case study for the GUHRI system, a scenario was defined in which a user and the robot interact in a classroom. The robot is the lecturer, and the user is the student. The robot is a female named “Nadine.” Nadine can understand the 12 upper body gestures described in Sect. 2.4.2.1 and react to the users’ gestures accordingly. In our system, Nadine is humanlike and capable of reacting by combining body movement, facial expression, and verbal language. In this way, Nadine’s reactions provide the user with vivid feedback. Figure 2.14 shows some examples of Nadine’s body movements along with corresponding facial expressions. Nonverbal behaviors can help to structure the processing of verbal information as well as giving affective feedback during the interaction [6, 17]. Thus, body movements and facial expressions are expected to enhance the quality of the interaction with Nadine.

In this scenario, Nadine’s behaviors are triggered by the users’ body language. Her reactions are consistent with the defined scenario (see Table 2.1). Note that because it is difficult to fully describe the robot’s body actions, the robot’s movements and emotional display are described at a high level. All the 12 upper body gestures are involved. The GHURI system

can also handle unexpected situations during the interaction. For example, Nadine can react appropriately even if the user suddenly answers a coming phone call.

2.4.2.5 Experiment and Discussion

A human upper body gesture dataset was built to test the proposed gesture recognition method. This dataset involves all the 12 upper body gestures mentioned in Sect. 2.4.2.1. The samples are captured from 25 volunteers of different genders,

Table 2.1 The scenario for human–robot interaction

Human gestures	Nadine’s response	
	Nonverbal	Verbal
“be confident”	Happy	It is great to see you so confident
“have question”	Moderate	What is your question?
“object”	Sad	Why do you disagree?
“praise”	Happy	Thank you for your praise
“stop”	Moderate	Why do you stop me?
“succeed”	Happy	Well done. You are successful
“shake hand”	Happy	Nice to meet you
“weakly agree”	Head nod	OK, we finally reach an agreement
“call”	Head shake	Please turn off your phone
“drink”	Moderate	You can have a drink. No problem
“read”	Moderate	Please, take your time and read it carefully
“write”	Moderate	If you need time for taking notes, I can slow my presentation

body sizes, and races. During the sample collection, no strict constraint was imposed on the people. They carried out the gestures based on their own habits. The user– Kinect relative position was also not strictly limited. For convenience, CyberGlove II was precalibrated for all the people with a standard calibration. Due to the dataset collection setup, large diversities may exist among the gesture samples from dif- ferent people. This will yield challenges on body gesture recognition. Figure 2.15 exhibits parts of the *Category 1* and *Category 2* gesture samples (“have question,” “succeed,” “call,” and “drink”) captured from five people for comparison. For the sake of brevity, not all the gestures are shown. The five descriptive and reference skeletal joints proposed in Sect. 2.4.2.2 are marked as big dots in Fig. 2.15, and they are connected by the straight segments to shape the upper body posture intuitively. From the exhibited samples, we can observe that:

- For the different people, the listed body gestures can indeed be differentiated from the hand and upper body posture information, and

the people execute the gestures differently to a certain degree. As aforementioned, this phenomenon leads to challenges on upper body gesture recognition;

- For different people and gestures, the five skeletal joints employed for gesture recognition can be tracked robustly, even when human–object interaction occurs. Generally, their resulting positions are accurate for gesture recognition. Mean- while, CyberGlove II is a human-touch device that can capture the hand posture

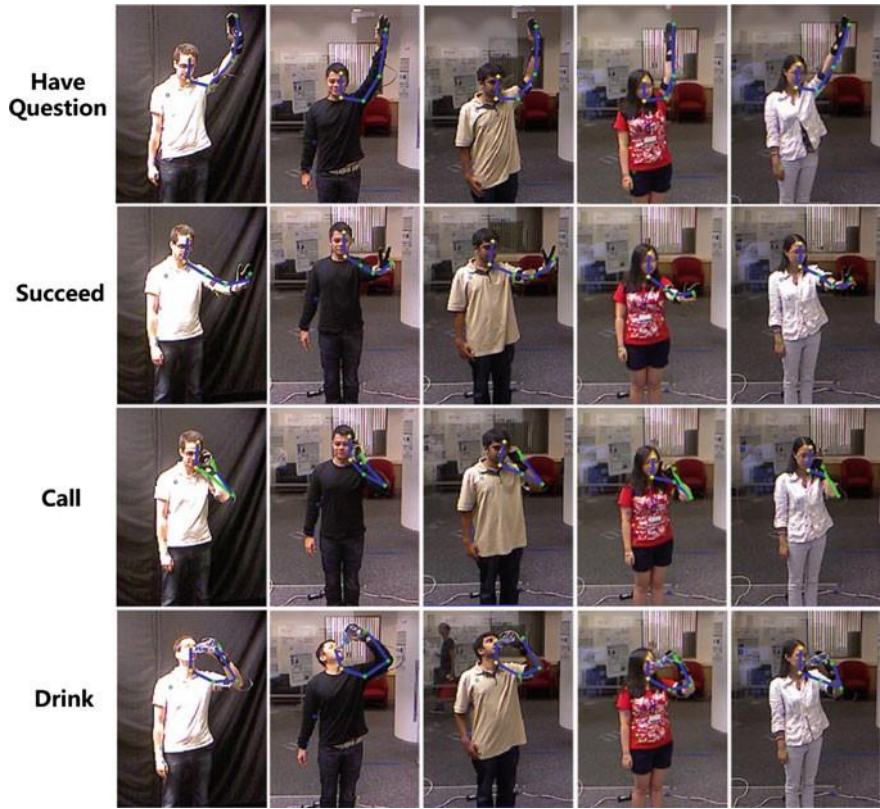


Fig. 2.15 Some gesture samples captured from different volunteers. These people are of different genders, body sizes, and races. They executed the gestures based on their own habits

robustly to yield high-accuracy data. Thus, the proposed human gesture-capture sensor solution can stably acquire available data for gesture recognition.

For each gesture, one key snapshot is picked up to build the dataset among all the 25 people. As a consequence, the resulting dataset contains 25 12 300 gesture samples in all. During experiment, the samples are randomly split into the training and testing set five times, and the average classification accuracy and standard deviation are reported.

The KNN classifier is used as the baseline to make comparison with the energy- based LMNN classifier. They are compared both on the items of classification accu- racy and on time consumption. The KNN classifier runs with different kinds of distance measures. Following the experimental

setup in [33], " k " is set as 3 in all cases. As the training sample number is a crucial factor that affects the classification accuracy, the results of two classifiers are compared corresponding to different amounts of training samples. For each class, the training sample number will increase from 4 to 14 with step size 2.

Table 2.2 Classification result (%) of the constructed *upper* body gesture dataset

Classifiers	Training sample number per class		
	4	6	8
KNN (Euclidean)	86.51(±2.89)	89.56(±2.43)	91.47(±2.21)
KNN (PCA)	73.81(±4.78)	84.04(±5.41)	79.31(±3.09)
KNN (LDA)	79.68(±5.33)	90.44(±2.56)	92.35(±2.44)
KNN (LMNN)	86.67(±2.18)	90.35(±2.56)	92.16(±1.80)
Energy (LMNN)	90.00(±3.40)	92.28(±0.85)	94.31(±2.04)
	10	12	14
KNN (Euclidean)	93.00(±1.15)	93.33(±0.73)	92.27(±2.30)
KNN (PCA)	86.11(±2.75)	88.21(±4.17)	86.67(±3.70)
KNN (LDA)	92.44(±1.34)	94.74(±0.84)	93.48(±1.38)
KNN (LMNN)	93.67(±1.34)	93.85(±1.48)	93.48(±1.15)
Energy (LMNN)	95.22(±1.50)	95.64(±1.39)	96.52(±2.37)

The best performance is shown in boldface. Standard deviations are in parentheses

Other two well-known distance metric learning methods, PCA [16] and LDA [12], are used for comparison with the LMNN distance metric learning approach. For PCA, the first 10 eigenvectors are used to capture roughly 90 % of the sum of eigenvalues, while the first 6 eigenvectors are used for LDA. The distance measures yielded by PCA and LDA are applied to the KNN classifier.

Table 2.2 lists the classification results yielded by the different classifier and distance measure combinations. It can be observed that:

- The 12 upper body gestures in the dataset can be well recognized by the proposed gesture recognition method. More than 95.00 % classification accuracy can be achieved if enough training samples are used. With the increase in training sample amount, the performance is generally enhanced consistently;
- Corresponding to all the training sample numbers, the energy-based LMNN classifier can yield the highest classification accuracy. Even with a small number (such as 4) of training samples it can still achieve relative good performance (90.00 %). When the training sample number reaches 14, the classification accuracy (96.52 %) is nearly satisfied for practical use, and its standard deviations are relatively low in most cases, which means

that the energy-based LMNN classifier is robust to the gesture diversities among people;

- KNN classifier can also yield good results on this dataset. However, it is inferior to the energy-based LMNN classifier. Compared to Euclidean distance, LMNN distance metric learning method can improve the performance of KNN classifier consistently in most cases. However, it works much better on the energy-based model;
- PCA does not work well on this dataset. Its performance is worse than the basic Euclidean distance. The reason may be that PCA needs a large number of training

Table 2.3 Average testing time consumption (ms) per sample

Classifiers	Training sample number per class		
	4	6	8
KNN (LDA)	0.0317	0.0398	0.0414
KNN (LMNN)	0.0239	0.0282	0.0328
Energy (LMNN)	0.0959	0.1074	0.1273
	10	12	14
KNN (LDA)	0.0469	0.0498	0.0649
KNN (LMNN)	0.0342	0.0418	0.0525
Energy (LMNN)	0.1359	0.1610	0.1943

The program is run on the computer with Intel (R) Core (TM) i5-2430M @ 2.4 GHz (only using one core)

samples to obtain the satisfied distance measures [22]. This is the limitation for practical applications.

- LDA also achieves good performance for upper body gesture recognition. However, it is still consistently inferior to energy-based LMNN, especially when the training sample number is small. For example, when the training sample number is only 4, energy-based LMNN’s accuracy (90.00 %) is significantly better than that of LDA (79.68 %) by a large margin (10.32 %).

Besides the classification accuracy, the testing time consumption is also what we are concerned about. The reason is that the GUHRI system should run in real-time for good HRI experience. According to the classification results in Table 2.2, the energy-based LMNN classifier, LMNN KNN classifier, and LDA KNN classifier are the three strongest ones for gesture recognition. Here, comparison on their testing time is also made. Table 2.3 lists the average running time per testing sample of the three classifiers, corresponding to different amounts of training samples. We can see that the three classifiers are extremely fast under our experimental conditions, and the time consumption mainly depends on the number of training samples. Frankly, the LDA KNN classifier and LMNN KNN classifier are much faster than the energy- based LMNN classifier. If a huge number of training samples were used (such as tens of thousands), the LDA KNN classifier and LMNN KNN classifier would be a better choice to achieve the balance between classification accuracy and computational efficiency.

2.5 Future Avenues

Our current research mainly pays attention to understanding the static upper body gestures. However, the dynamic ones are also the essential components of body language in daily life. For human–human interaction, they provide additional communication clues to boost the interaction performance. To make human–robot interaction

more natural and lifelike, the robot should be capable of understanding both static and dynamic gestures. Being different from static gestures, motion information is required to recognize the dynamic gestures. From Microsoft Kinect SDK, human body 3D joint information can be achieved in real-time (30fps). Motion information can be intuitively extracted from the position change in body joints along the temporal axis. The body joint motion information has earlier been applied to activity recognition [31, 32]. Nevertheless, these works still ignore hand gestures, which may lead to ambiguity on gesture understanding. Thus, one of our future research avenues is to recognize the dynamic human upper body gestures (such as “wave hand,” “say no,” and “clap,” etc.) by combining both body motion and hand gesture information.

Another future research topic is to recognize human gestures from the egocentric perspectives of the robot. In the proposed GUHRI system, Kinect is employed as the vision sensor with fixed position. This system setup has some limits for real applications under challenging conditions. That is, the robot cannot change her viewpoint due to the fixed position of Kinect. In this case, the robot is not able to always acquire the optimal viewpoint to capture human body gesture information. Actually, due to this viewpoint reason, body occlusion may happen that will seriously confuse the accurate body joint position extraction. One feasible solution for this problem is to capture the human body gesture information from the robot’s egocentric perspectives. In this way, the robot can change her viewpoint accordingly. However, to achieve good results, some new challenges need to be solved; one main problem is how to distinguish camera motion and real body motion.

In addition, how to integrate the verbal clues in the GURHI system to further enhance the human–robot interaction performance is also what we are concerned about in the future work. Making the robot “see” and “listen” will let her become more autonomous and humanlike.

2.6 Conclusion

The GUHRI system, a novel body gesture understanding and human–robot interaction system, is proposed in this paper. A set of 12 human upper body gestures with and without human–object interactions can be understood by the robot. Meanwhile, the robot can express herself by using a combination of body movements, facial expressions, and verbal language simultaneously, aiming to give the users a natural and vivid experience.

A new combination of sensors is proposed. That is, CyberGlove II and Kinect are combined to capture the head, arm, and hand posture simultaneously. An effective and real-time gesture recognition method is also proposed. In the experiment, a human upper body gesture dataset is built. The experimental results demonstrate the effectiveness and efficiency of our gesture recognition method.

So far, the gestures involved in GUHRI system have been static ones, e.g., “*have question*,” “*praise*,” “*call*,” “*drink*,” etc. As the future work, we plan to enable the robot to understand dynamic gestures such as “*wave hand*,” “*say no*,” “*clap*,” etc. Speech recognition can be further added to make the interaction more natural.

References

1. Adamo-Villani N, Heisler J, Arns L (2007) Two gesture recognition systems for immersive math education of the deaf. In: Proceedings of the first international conference on immersive telecommunications (ICIT 2007). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), p 9
2. Beck A, Cañamero L, Hiole A, Damiano L, Cosi P, Tesser F, Somavilla G (2013) Interpretation of emotional body language displayed by a humanoid robot: a case study with children. *Int J Soc Robot*: 1–10
3. Belongies S, Malik J, Puzicha J (2002) Shape matching and object recognition using shape contexts. *IEEE Trans Pattern Anal Mach Int* 24(4):509–522
4. Berman S, Stern H (2012) Sensors for gesture recognition systems. *IEEE Trans Syst Man Cybern: Appl Rev* 42(3):277–290
5. Brethes L, Menezes P, Lerasle F, Hayet J (2004) Face tracking and hand gesture recognition for human-robot interaction. In: Proceedings of IEEE conference on robotics and automation (ICRA 2004), IEEE, vol 2. pp 1901–1906
6. Cañamero L, Fredslund J (2001) I show you how i like you—can you read it in my face? [robotics]. *IEEE Trans Syst Man Cybern: Syst Hum* 31(5):454–459
7. Cassell J et al. (2000) Nudge nudge wink wink: elements of face-to-face conversation for embodied conversational agents. *Embodied conversational agents*, pp 1–27
8. Chopra S, Hadsell R, LeCun Y (2005) Learning a similarity metric discriminatively, with application to face verification. In: Proceedings of IEEE conference on computer vision and pattern recognition (CVPR 2005), vol 1. pp 539–546
9. Cover T, Hart P (1967) Nearest neighbor pattern classification. *IEEE Trans Inf Theory* 13(1):21–27
10. Dautenhahn K (2007) Socially intelligent robots: dimensions of human-robot interaction. *Philos Trans Royal Soc B: Biol Sci* 362(1480):679–704
11. Faber F, Bennewitz M, Eppner C, Gorog A, Gonsior C, Joho D, Schreiber M, Behnke S (2009) The humanoid museum tour guide robotinho. In: Proceedings of IEEE symposium on robot and human interactive communication (RO-MAN 2009), IEEE, pp 891–896
12. Fisher RA (1936) The use of multiple measures in taxonomic problems. *Ann Eugenics* 7:179–188
13. Fong T, Nourbakhsh I, Dautenhahn K (2003) A survey of socially interactive robots. *Robot Auton Syst* 42(3):143–166
14. Goodrich MA, Schultz AC (2007) Human-robot interaction: a survey. *Found Trends Hum-Comput Interact* 1(3):203–275
15. Immersion (2010) Cyberglove II specifications
16. Jolliffe IT (1986) Principal component analysis. Springer, London

17. Krämer NC, Tietz B, Bente G (2003) Effects of embodied interface agents and their gestural activity. In: *Intelligent virtual agents*. Springer, London, pp 292–300
18. Lu G, Shark L-K, Hall G, Zeshan U (2012) Immersive manipulation of virtual objects through glove-based hand gesture interaction. *Virtual Reality* 16(3):243–252
19. Mehrabian A (1971) *Silent messages*
20. Müller M, Röder T (2006) Motion templates for automatic classification and retrieval of motion capture data. In: *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on computer animation (SCA 2006)*, Eurographics Association, pp 137–146

21. Nickel K, Stiefelbogen R (2007) Visual recognition of pointing gestures for human-robot interaction. *Image Vis Comput* 25(12):1875–1884
22. Osborne JW, Costello AB (2004) Sample size and subject to item ratio in principal components analysis. *Pract Assess, Res Eval* 9(11):8
23. Perzanowski D, Schultz AC, Adams W, Marsh E, Bugajska M (2001) Building a multimodal human-robot interface. *IEEE Intell Syst* 16(1):16–21
24. Shotton J, Fitzgibbon A, Cook M, Sharp T, Finocchio M, Moore R, Kipman A, Blake A (2011) Real-time human pose recognition in parts from single depth images. In: *Proceedings of IEEE conference on computer vision and pattern recognition (CVPR 2011)*, pp 1297–1304
25. Smith LB, Breazeal C (2007) The dynamic lift of developmental process. *Dev Sci* 10(1):61–68
26. Spiliotopoulos D, Androutsopoulos I, Spyropoulos CD (2001) Human-robot interaction based on spoken natural language dialogue. In: *Proceedings of the European workshop on service and humanoid robots*, pp 25–27
27. Stiefelbogen R, Ekenel HK, Fugen C, Gieselmann P, Holzapfel H, Kraft F, Nickel K, Voit M, Waibel A (2007) Enabling multimodal human-robot interaction for the karlsruhe humanoid robot. *IEEE Trans Robot* 23(5):840–851
28. Stiefelbogen R, Fugen C, Gieselmann R, Holzapfel H, Nickel K, Waibel A (2004) Natural human-robot interaction using speech, head pose and gestures. In: *Proceedings of IEEE conference on intelligent robots and systems (IROS 2004)*, IEEE, vol 3, pp 2422–2427
29. Teleb H, Chang G (2012) Data glove integration with 3d virtual environments. In: *Proceedings of international conference on systems and informatics (ICSAI 2012)*, IEEE, pp 107–112
30. Waldherr S, Romero R, Thrun S (2000) A gesture based interface for human-robot interaction. *Auton Robots* 9(2):151–173
31. Wang J, Liu Z, Wu Y, Yuan J (2012) Mining actionlet ensemble for action recognition with depth cameras. In: *Proceedings of IEEE conference on computer vision and pattern recognition (CVPR 2012)*, pp 1290–1297
32. Wang J, Liu Z, Chorowski J, Chen Z, Wu Y (2012) Robust 3d action recognition with random occupancy patterns. In: *Proceedings of European conference on computer vision (ECCV)*. Springer, London, pp 872–885
33. Weinberger KQ, Saul LK (2009) Distance metric learning for large margin nearest neighbor classification. *J Mach Learn Res* 10:207–244
34. Xiao Y, Cao Z, Zhuo W (2011) Type-2 fuzzy thresholding using glsc histogram of human visual nonlinearity characteristics. *Opt. Express* 19(11):10656–10672
35. Xiao Y, Cao Z, Yuan J (2014) Entropic image thresholding based on GLGM histogram. *Pattern Recogn Lett* 40:47–55
36. Xiao Y, Wu J, Yuan J (2014) mCENTRIST: a multi-channel feature generation mechanism for scene categorization. *IEEE Trans Image Process* 23(2):823–836
37. Xiao Y, Yuan J, Thalmann D (2013) Human-virtual human interaction by upper body gesture understanding. In: *Proceedings of the 19th ACM symposium on virtual reality software and technology (VRST 2013)*, pp 133–142. ACM, Las Vegas
38. Xiao Y, Zhang Z, Beck A, Yuan J, Thalmann D (2014) Human-robot interaction by understanding upper body gestures. *Presence: teleoperators and virtual environments* (Accepted)

5. Research Methodology

II

Empirical and Statistical Approaches

7	Corpus Creation <i>Richard Xiao</i>	147
	Introduction · Corpus Size · Balance, Representativeness, and Sampling · Data Capture and Copyright · Corpus Markup and Annotation · Multilingual Corpora · Multimodal Corpora · Conclusions · References	
8	Treebank Annotation <i>Eva Hajičová, Anne Abeillé, Jan Hajič, Jiří Mírovský, and Zdeňka Urešová</i>	167
	Introduction · Corpus Annotation Types · Morphosyntactic Annotation · Treebanks: Syntactic, Semantic, and Discourse Annotation · The Process of Building Treebanks · Applications of Treebanks · Searching Treebanks · Conclusions · Acknowledgments · References	
9	Fundamental Statistical Techniques <i>Tong Zhang</i>	189
	Binary Linear Classification · One-versus-All Method for Multi-Category Classification · Maximum Likelihood Estimation · Generative and Discriminative Models · Mixture Model and EM · Sequence Prediction Models · References	
10	Part-of-Speech Tagging <i>Tunga Güngör</i>	205
	Introduction · The General Framework · Part-of-Speech Tagging Approaches · Other Statistical and Machine Learning Approaches · POS Tagging in Languages Other Than English · Conclusion · References	
11	Statistical Parsing <i>Joakim Nivre</i>	237
	Introduction · Basic Concepts and Terminology · Probabilistic Context-Free Grammars · Generative Models · Discriminative Models · Beyond Supervised Parsing · Summary and Conclusions · Acknowledgments · References	

12 Multiword Expressions *Timothy Baldwin and Su Nam Kim*.....267

Introduction · Linguistic Properties of MWEs · Types of MWEs · MWE Classification ·

Research Issues · Summary · Acknowledgments · References

Corpus Creation

7.1	Introduction	147
7.2	Corpus Size	148
7.3	Balance, Representativeness, and Sampling.....	149
7.4	Data Capture and Copyright	153
7.5	Corpus Markup and Annotation.....	155
7.6	Multilingual Corpora.....	159
7.7	Multimodal Corpora.....	161
7.8	Conclusions.....	161
	References	162

Richard Xiao
Edge Hill University

7.1 Introduction

A corpus can be defined as a collection of machine-readable authentic texts (including transcripts of spoken data) that is sampled to be representative of a particular natural language or language variety (McEnery et al. 2006: 5), though “representativeness” is a fluid concept (see Section 7.3). Corpora play an essential role in natural language processing (NLP) research as well as a wide range of linguistic investigations. They provide a material basis and a test bed for building NLP systems. On the other hand, NLP research has contributed substantially to corpus development (see Dipper 2008 for a discussion of the relationship between corpus linguistics and computational linguistics), especially in corpus annotation, for example, part-of-speech tagging (see Chapter 10), syntactic parsing (see Chapters 8 and 11), semantic tagging (see Chapters 5 and 14), as well as the alignment of parallel corpora (see Chapter 16).

There are thousands of corpora in the world, but most of them are created for specific research projects and are not publicly available. Xiao (2008) provides a comprehensive survey of a wide range of well-known and influential corpora in English and many other languages, while a survey of corpora for less-studied languages can be found in Ostler (2008). Since corpus creation is an activity that takes time and costs money, it is certainly desirable for readers to use such ready-made corpora to carry out their work. Unfortunately, however, this is not always feasible or possible. As a corpus is always designed for a particular purpose, the usefulness of a ready-made corpus must be judged with regard to the purpose to which a user intends to put it. Consequently, while there are many corpora readily available, it is often the case that readers will find that they are not able to address their research questions using ready-made corpora. In such circumstances, one must build one’s own corpus. This chapter covers principal considerations involved in creating such DIY (“do-it-yourself”) corpora as well as the issues that come up in major corpus creation projects.

This chapter discusses core issues in corpus creation such as corpus size, representativeness, balance

and sampling, data capture and copyright, markup and annotation, as well as peripheral issues such as multilingual and multimodal corpora.

7.2 Corpus Size

One must be clear about one's research question (or questions) when planning to build a DIY corpus. This helps you to determine what material you will need to collect. For example, if you wish to compare British English and American English, you will need to collect spoken and/or written data produced by native speakers of the two regional varieties of English; if you are interested in how Chinese speakers acquire French as a second language, you will then need to collect the French data produced by Chinese learners to create a learner corpus; if you are interested in how the English language has evolved over centuries, you will need to collect samples of English produced in different historical periods to build a historical or diachronic corpus. Readers are reminded, though, that many corpora of these kinds are now already available (see Xiao 2008 for a recent survey). Having developed an understanding of the type of data you need to collect, and having made sure that no ready-made corpus of such material exists, one needs to find a source of data. Assuming that the data can be found, one then has to address the question of corpus size.

How large a corpus do you need? There is no easy answer to this question. The size of the corpus needed depends upon the purpose for which it is intended as well as a number of practical considerations. In the early 1960s, when the processing power and storage capacity of computers were quite limited, a one-million-word corpus such as the Brown corpus (i.e., the Brown University Standard Corpus of Present-day American English, see Kučera and Francis 1967) appeared to be as large a corpus as one could reasonably build. With the increase in computer power and the availability of machine-readable texts, however, a corpus of this size is no longer considered large, and in comparison with today's giant corpora like the 100-million-word British National Corpus (BNC, see Aston and Burnard 1998) and the 524-million-word Bank of English (BoE, Collins 2007) it appears somewhat small. An interesting discussion of corpus size and design can be found in Keller and Lapata (2003), who compare similarities and differences in the frequencies for bigrams (i.e., two-word clusters) obtained from the BNC and the Web.

The availability of suitable data, especially in machine-readable form, seriously affects corpus size. In building a *balanced* corpus according to fixed proportions (see Section 7.3), for example, the lack of data for one text type may accordingly restrict the size of the samples of other text types taken. This is especially the case for parallel corpora, as it is common for the availability of translations to be unbalanced across text types for many languages. For example, it will be much easier to find Chinese translations of English news stories than English translations of Chinese literary texts. While it is often possible to transfer paper-based texts into electronic form using OCR (optical character recognition) software, the process costs time and money and is error-prone. Hence, the availability of machine-readable data is often the main limiting factor in corpus creation.

Another factor that potentially limits the size of a DIY corpus is copyright (see Section 7.4 for further discussion). Unless the proposed corpus contains entirely out-of-date or copyright-free data, simply gathering available data and using it in a freely available corpus may expose the corpus creator to legal action. When one seeks copyright clearance, one can face frustration—the construction of the corpus is your priority, not the copyright holder's. They may simply ignore you. Their silence cannot be taken as consent. Copyright clearance in building a large corpus necessitates much effort, trouble, and frustration.

No matter how important legal considerations may seem, one should not lose sight of the paramount importance of the research question. This question controls all of your corpus-building decisions, including the decision regarding corpus size. Even if the conditions discussed above allow for a large corpus, it does not mean that a large corpus is what you want. First, the size of the corpus needed to explore a research question is dependent on the frequency and distribution of the linguistic features under consideration in that corpus (cf. McEnery and Wilson 2001: 80). As Leech (1991: 8–29) observes, size is not all-important. Small corpora may contain sufficient examples of frequent linguistic features. To study features such as

the number of present and past tense verbs in English, for example, a sample of 1000 words may prove

sufficient (Biber 1993). Second, small specialized corpora serve a very different yet important purpose from large multi-million-word corpora (Shimazumi and Berber-Sardinha 1996). It is understandable that corpora for lexical studies are much larger than those for grammatical studies, because when studying lexis one is interested in the frequency of the distribution of a word (see Baroni 2009 for a discussion of distributions in text), which can be modeled as contrasting with all others of the same category (cf. Santos 1996:11). In contrast, corpora employed in quantitative studies of grammatical devices can be relatively small (cf. Biber 1988; Givon 1995), because the syntactic freezing point is fairly low (Hakulinen et al. 1980: 104). Third, corpora that need extensive manual annotation (e.g., pragmatic annotation) are necessarily small. Fourth, many corpus tools set a ceiling on the number of concordances that can be extracted, for example, WordSmith version 3.0 can extract a maximum of 16,868 concordances (versions 4.0 and 5.0 do not have this limit). This makes it inconvenient for a frequent linguistic feature to be extracted from a very large corpus. Even if this can be done, few researchers can obtain useful information from hundreds of thousands of concordances (cf. Hunston 2002: 25). The data extracted defies manual analysis by a sole researcher by virtue of the sheer volume of examples discovered. Of course, I do not mean that DIY corpora must necessarily be small. A corpus small enough to produce only a dozen concordances of a linguistic feature under consideration will not be able to provide a reliable basis for quantification, though it may act as a spur to qualitative research.

It is important to note, however, that corpus size is an issue of ongoing debate in corpus creation. Some corpus linguists have argued that size matters (e.g., Krishnamurthy 2000; Sinclair 2004; Granath 2007). Large corpora are certainly of advantage in lexicography and in the study of infrequent linguistic structures (e.g., Keller and Lapata 2003). Also, NLP and language engineering can have different requirements for corpora from those used in linguistic research as discussed above. Corpora used in NLP and language engineering tend to be domain- or genre-specific specialized corpora (e.g., those composed of newspapers or telephone-based transactional dialogues), data for which are often easier to collect in large amounts than for balanced corpora. Furthermore, larger corpora are more reliable in statistical modeling, which is essential in natural language processing and language engineering. In a word, the point I wish to make is that the optimum size of a corpus is determined by the research question the corpus is intended to address as well as practical considerations.

7.3 Balance, Representativeness, and Sampling

One of the commonly accepted defining features of a corpus, which distinguishes a corpus from an archive (i.e., a random collection of texts), is *representativeness*. A corpus is designed to represent a particular language or language variety whereas an archive is not. What does representativeness mean in corpus linguistics? According to Leech (1991: 27), a corpus is thought to be representative of the language variety it is supposed to represent if the findings based on its contents can be generalized to the said language variety. Biber (1993: 243) defines representativeness from the viewpoint of how this quality is achieved: "Representativeness refers to the extent to which a sample includes the full range of variability in a population." A corpus is essentially a sample of a language or language variety (i.e., population). Sampling is entailed in the creation of virtually any corpus of a living language. In this respect, the representativeness of most corpora is to a great extent determined by two factors: the range of *genres, domains, and media* included in a corpus (i.e., *balance*) and how the text chunks for each genre are selected (i.e., *sampling*).

The criteria used to select texts for inclusion in a corpus are principally external to the texts themselves and dependent upon the intended use for the corpus (Aston and Burnard 1998: 23). The distinction between external and internal criteria corresponds to Biber's (1993: 243) situational vs. linguistic perspectives. External criteria are defined situationally irrespective of the distribution of linguistic features whereas internal criteria are defined linguistically, taking into account the distribution of such features. Internal criteria have sometimes been proposed as a measure of corpus representativeness (e.g., Otlo-

getswe 2004). In my view, it is problematic; indeed it is circular, to use internal criteria such as the

distribution of words or grammatical features as the primary parameters for the selection of corpus data. A corpus is typically designed to study linguistic distributions. If the distribution of linguistic features is predetermined when the corpus is designed, there is no point in analyzing such a corpus to discover naturally occurring linguistic feature distributions. The corpus has been skewed by design. As such, I agree with Sinclair (2005) when he says that the texts or parts of texts to be included in a corpus should be selected according to external criteria so that their linguistic characteristics are, initially at least, independent of the selection process. This view is also shared by many other scholars including Atkins et al. (1992: 5–6) and Biber (1993: 256). Yet, once a corpus is created by using external criteria, the results of corpus analysis can be used as feedback to improve the representativeness of the corpus. In Biber's (1993: 256) words, "the compilation of a representative corpus should proceed in a cyclical fashion."

In addition to text selection criteria, Hunston (2002: 30) suggests that another aspect of representativeness is change over time: "Any corpus that is not regularly updated rapidly becomes unrepresentative." The relevance of permanence in corpus design actually depends on how we view a corpus, that is, whether a corpus should be viewed as a static or dynamic language model. The static view typically applies to a *sample corpus* whereas a dynamic view applies to a *monitor corpus*. A monitor corpus is primarily designed to track changes from different periods (cf. Hunston 2002: 16). It is particularly useful in tracking relatively rapid language change, such as the development and the life cycle of neologisms. Monitor corpora are constantly (e.g., annually, monthly, or even daily) supplemented with fresh material and keep increasing in size. For example, the Bank of English (BoE) has increased in size progressively since its inception in the 1980s (Hunston 2002: 15) and is around 524 million words at present. In contrast, a sample corpus is designed to represent a static snapshot of a particular language variety at a particular time. Static sample corpora, if resampled, may also allow the study of slower paced language change over time. For example, the LOB (Lancaster-Oslo-Bergen Corpus of British English, Johansson et al. 1978) and Brown corpora are supposed to represent written British and American English in the early 1960s; and their recent updates, Freiberg-LOB (FLOB, see Hundt et al. 1998) and Freiberg-Brown (Frown, see Hundt et al. 1999) corpora, represent written British and American English in the early 1990s respectively. Sample corpora such as these make it possible to track language change over the intervening three decades.

In addition to the distinction between sample and monitor corpora, representativeness has different meanings for *general* and *specialized* corpora. Corpora of the first type typically serve as a basis for an overall description of a language or language variety. The BNC corpus, for example, is supposed to represent modern British English as a whole. In contrast, a specialized corpus tends to be specific to a particular domain (e.g., medicine or law) or genre (e.g., newspaper text or academic prose). For a general corpus, it is understandable that it should cover, proportionally, as many text types as possible so that the corpus is maximally representative of the language or language variety it is supposed to represent. Even a specialized corpus, for example, one dealing with telephone calls to an operator service should be balanced by including within it a wide range of types of operator conversations (e.g., line fault, request for an engineer call out, number check, etc.) between a range of operators and customers (cf. McEnery et al. 2001) so that it can be claimed to represent this variety of language.

While both general and specialized corpora should be representative of a language or language variety, they have different criteria for representativeness. The representativeness of a general corpus depends heavily on sampling from a broad range of genres whereas the representativeness of a specialized corpus, at the lexical level at least, can be measured by the degree of *closure* (McEnery and Wilson 2001: 166) or *saturation* (Belica 1996: 61–74) of the corpus. Closure/saturation for a particular linguistic feature (e.g., size of lexicon) of a variety of language (e.g., computer manuals) means that the feature appears to be finite or is subject to very limited variation beyond a certain point. To measure the saturation of a corpus, the corpus is first divided into segments of equal size based on its tokens. The corpus is said to be saturated at the lexical level if each addition of a new segment yields approximately the same number of new lexical items as the previous segment, that is, when the curve of lexical growth is asymptotic, or flattening out. The notion of saturation is claimed to be superior to such concepts as balance for its

measurability (Teubert 2000). It should be noted, however, that saturation is only concerned with lexical

features. While it may be possible to adapt saturation to measure features other than lexical growth, there have been few attempts to do this to date (though see McEnery and Wilson 2001: 176–183 for a study of part-of-speech and sentence type closure).

It appears, then, that the representativeness of a corpus, especially a general corpus, depends primarily on how balanced the corpus is; in other words, the range of text categories included in the corpus. As with representativeness, the acceptable balance of a corpus is determined by its intended uses. Hence, a general corpus that contains both written and spoken data (e.g., the BNC) is balanced; so are written corpora such as Brown and LOB, and spoken corpora such as the Cambridge Nottingham Corpus of Discourse in English (CANCODE). A balanced corpus usually covers a wide range of text categories that are supposed to be representative of the language or language variety under consideration. These text categories are typically sampled proportionally for inclusion in a corpus so that “it offers a manageably small scale model of the linguistic material which the corpus builders wish to study” (Atkins et al. 1992: 6).

Balance appears to be a more important issue for a static sample corpus than for a dynamic monitor corpus. As corpora of the latter type are updated frequently, it is usually “impossible to maintain a corpus that also includes text of many different types, as some of them are just too expensive or time consuming to collect on a regular basis” (Hunston 2002: 30–31). The builders of monitor corpora appear to feel that balance has become less of a priority — sheer size seems to have become the basis of the corpus’s authority, under the implicit and arguably unwarranted assumption that a corpus will in effect balance itself when it reaches a substantial size.

While balance and representativeness are important considerations in corpus design, they depend on the research question and the ease with which data can be captured and thus must be interpreted in relative terms. In other words, a corpus should only be as representative as possible of the language variety under consideration. For example, if one wants a corpus that is representative of general English, a corpus representative of newspapers will not do; if one wants a corpus representative of newspapers, a corpus representative of *The Times* will not do. Corpus balance and representativeness are fluid concepts that link directly to research questions. The research question one has in mind when building (or thinking of using) a corpus defines the required balance and representativeness. Any claim of corpus balance is largely an act of faith rather than a statement of fact as, at present, there is no reliable scientific measure of corpus balance. Rather the notion relies heavily on intuition and best estimates. Another argument supporting a loose interpretation of balance and representativeness is that these notions *per se* are open to question (cf. Hunston 2002: 28–30). To achieve corpus representativeness along the lines of the Brown corpus model one must know how often each genre is used by the language community in the *sampling period*. Yet it is unrealistic to determine the correlation of language production and reception in various genres (cf. Hausser 1999: 291; Hunston 2002: 29). The only solution to this problem is to treat corpus-based findings with caution. It is advisable to base your claims on your corpus and avoid unreasonable generalizations. Likewise, conclusions drawn from a particular corpus must be treated as deductions rather than facts (cf. also Hunston 2002: 23). With that said, however, I entirely agree with Atkins et al. (1992: 6), who comment that:

It would be short-sighted indeed to wait until one can scientifically balance a corpus before starting to use one, and hasty to dismiss the results of corpus analysis as “unreliable” or “irrelevant” because the corpus used cannot be proved to be ‘balanced.’

Given that language is infinite whereas a corpus is finite in size, *sampling* is unavoidable in corpus creation. Unsurprisingly, corpus representativeness and balance are closely associated with sampling. Given that we cannot exhaustively describe natural language, we need to sample it in order to achieve a level of balance and representativeness that matches our research question. Having decided that sampling is inevitable, there are important decisions that must be made about how to sample so that the resulting corpus is as balanced and representative as practically possible.

As noted earlier, with few exceptions, a corpus is typically a sample of a much larger *population*.

A sample is assumed to be representative if what we find for the sample also holds for the general

population (cf. Manning and Schütze 1999: 119). In the statistical sense, samples are scaled down versions of a larger population (cf. Váradi 2000). The aim of sampling theory “is to secure a sample which, subject to limitations of size, will reproduce the characteristics of the population, especially those of immediate interest, as closely as possible” (Yates 1965: 9).

In order to obtain a representative sample from a population, the first concern to be addressed is to define the *sampling unit* and the boundaries of the population. For written text, for example, a sampling unit may be a book, a periodical, or a newspaper. The population is the assembly of all sampling units while the list of sampling units is referred to as a *sampling frame*. The population from which samples for the pioneering Brown corpus were drawn, for instance, was all written English text published in the United States in 1961 while its sampling frame was a list of the collection of books and periodicals in the Brown University Library and the Providence Athenaeum. For the LOB corpus, the target population was all written English text published in the United Kingdom in 1961 while its sampling frame included the *British National Bibliography Cumulated Subject Index* 1960–1964 for books and *Willing’s Press Guide* 1961 for periodicals.

In corpus design, a population can be defined in terms of language production, language reception, or language as a product. The first two designs are basically demographically oriented as they use the demographic distribution (e.g., age, sex, social class) of the individuals who produce/receive language data to define the population while the last design is organized around text category/genre of language data. As noted earlier, the Brown and LOB corpora were created using the criterion of language as a product while the BNC defines the population primarily on the basis of both language production and reception. However, it can be notoriously difficult to define a population or construct a sampling frame, particularly for spoken language, for which there are no ready-made sampling frames in the form of catalogues or bibliographies.

Once the target population and the sampling frame are defined, different sampling techniques can be applied to choose a sample that is as representative as possible of the population. A basic sampling method is *simple random sampling*. With this method, all sampling units within the sampling frame are numbered and the sample is chosen by use of a table of random numbers. As the chance of an item being chosen correlates positively with its frequency in the population, simple random sampling may generate a sample that does not include relatively rare items in the population, even though they can be of interest to researchers. One solution to this problem is *stratified random sampling*, which first divides the whole population into relatively homogeneous groups (so-called strata) and then samples each stratum at random (see Evert 2006 for a discussion of random sampling in corpus creation). In the Brown and LOB corpora, for example, the target population for each corpus was first grouped into 15 text categories such as news reportage, academic prose, and different types of fiction; samples were then drawn from each text category. Demographic sampling, which first categorizes sampling units in the population on the basis of speaker/writer age, sex and social class, is also a type of stratified sampling. Biber (1993) observes that a stratified sample is never less representative than a simple random sample.

A further decision to be made in sampling relates to *sample size*. For example, with written language, should we sample full texts (i.e., whole documents) or text chunks? If text chunks are to be sampled, should we sample text initial, middle, or end chunks? Full text samples are certainly useful in text linguistics, yet they may potentially constitute a challenge in dealing with vexatious copyright issues. Also, given its finite overall size, the coverage of a corpus including full texts may not be as balanced as a corpus including text segments of constant size. As a result, “the peculiarity of an individual style or topic may occasionally show through into the generalities” (Sinclair 1991: 19). Aston and Burnard (1998: 22) argue that the notion of “completeness” may sometimes be “inappropriate or problematic.” As such, unless a corpus is created to study such features as textual organization, or copyright holders have granted you permission to use full texts, it is advisable to sample text segments. According to Biber (1993: 252), frequent linguistic features are quite stable in their distributions and hence short text chunks (e.g., 2000 running words) are

usually sufficient for the study of such features while rare features are more varied in their distribution

and thus require larger samples (Baroni 2009). In selecting samples to be included in a corpus, however, attention must also be paid to ensure that text initial, middle, and end samples are balanced.

Another sampling issue, which particularly relates to stratified sampling, is the proportion and the number of samples for each text category. The numbers of samples across text categories should be proportional to their frequencies and/or weights in the target population in order for the resulting corpus to be considered as representative. Nevertheless, it has been observed that, as with defining a target population, such proportions can be difficult to determine objectively (cf. Hunston 2002: 28–30). Furthermore, the criteria used to classify texts into different categories or genres are often dependent on intuitions. As such, the representativeness of a corpus, as noted, should be viewed as a statement of belief rather than fact. In the Brown corpus, for example, a panel of experts determined the ratios between the 15 text categories. As for the number of samples required for each category, Biber (1993) demonstrates that ten 2000-word samples are typically sufficient.

The above discussion suggests that in creating a balanced, representative corpus, stratified random sampling is to be preferred over simple random sampling while different sampling methods should be used to select different types of data. For written texts, a text typology established on the basis of external criteria is highly relevant while for spoken data demographic sampling is appropriate. However, context-governed sampling must complement samples obtained from demographic sampling so that some contextually governed linguistic variations can be included in the resulting corpus.

7.4 Data Capture and Copyright

For pragmatic reasons noted in Section 7.2, electronic data is preferred over paper-based material in building DIY corpora. The World Wide Web (WWW) is an important source of machine-readable data for many languages. For example, digital text archives mounted on the Web such as Oxford Text Archive (<http://ota.ahds.ac.uk/>) and Project Gutenberg (<http://www.gutenberg.org/catalog/>) as well as the digital collections of some university libraries (e.g., <http://lib.virginia.edu/digital/collections/text/>, <http://onlinebooks.library.upenn.edu/>) provide large amounts of publicly accessible electronic texts.

The web pages on the Internet normally use Hypertext Markup Language (i.e., HTML) to enable browsers like Internet Explorer or Netscape to display them properly. While the tags (included in angled brackets) are typically hidden when a text is displayed in a browser, they do exist in the source file of a web page. Hence, an important step in building DIY corpora using web pages is tidying up the downloaded data by converting web pages to plain text, or to some desired format, for example, XML (see Section 7.5). In this section, I will introduce some useful tools to help readers to download data from the Internet and clean up the downloaded data by removing or converting HTML tags. These tools are either freeware or commercial products available at affordable prices.

While it is possible to download data page by page, which is rather time consuming, there are a number of tools that facilitate downloading all of the web pages on a selected Web site in one go (e.g., Grab-a-Site or HTTrack), or more usefully, downloading related web pages (e.g., containing certain key words) at one go. The WordSmith Tools (versions 4.0 and 5.0), for example, incorporates the WebGetter function that helps users to build DIY corpora. WebGetter downloads related Web pages with the help of a search engine (Scott 2003: 87). Users can specify the minimum file length or word number (small files may contain only links to a couple of pictures and nothing much else), required language and, optionally, required words. Web pages that satisfy the requirements are downloaded simultaneously (cf. Scott 2003: 88–89). The WebGetter function, however, does not remove the HTML markup or convert it to XML. The downloaded data needs to be tidied up using other tools before they can be loaded into a concordancer or further annotated.

Another tool worth mentioning is the freeware Multilingual Corpus Toolkit (MLCT, see Piao et al. 2002). The MLCT runs in Java Runtime Environment (JRE) version 1.4 or above, which is freely available

on the Internet. In addition to many other functions needed for multilingual language processing (e.g.,

markup, part-of-speech tagging, and concordancing), the system can be used to extract texts from the Internet. Once a webpage is downloaded, it is cleaned up. One weakness of the program is that it can only download one web page at a time. Yet this weakness is compensated for by another utility that converts all of the web pages in a file folder (e.g., the web pages downloaded using the Webgetter function of WordSmith version 4.0) to a desired text format in one go. Another attraction of the MLCT is that it can mark up textual structure (e.g., paragraphs and sentences) automatically.

Finally, the BootCaT Toolkit provides a suite of utilities that allow the user to bootstrap specialized corpora and terms from the Web on the basis of a small set of terms as input (Baroni and Bernardini 2004). Readers interested in the Web as corpus can refer to Kilgariff and Grefenstette (2003), Baroni and Bernardini (2006), and Hundt et al. (2007), and refer to Keller and Lapata (2003) for a comparison of the frequencies obtained from the Web and a balanced corpus such as the BNC.

A major issue in data collection is copyright. While it is possible to use copyright-free material in corpus creation, such data are usually old and a corpus consisting entirely of such data is not useful if one wishes to study contemporary English, for example. Such corpora are even less useful in NLP research, which tends to focus on current language use. Simply using copyrighted material in a corpus without the permission of the copyright holders may cause unnecessary trouble. In terms of purposes, corpora are typically of two types: for commercial purposes or for non-profit-making academic research. It is clearly unethical and illegal to use the data of other copyright holders to make money solely for oneself. Creators of commercial corpora usually reach an agreement with copyright holders as to how the profit will be shared. Publishers as copyright holders are also usually willing to contribute their data to a corpus-building project if they can benefit from the resulting corpus (e.g., the British National Corpus, the Longman Corpus Network, and the Cambridge International Corpus).

In creating DIY corpora for use in non-profit-making research, you might think that you need not worry about copyright if you are not selling your corpus to make a profit. Sadly, this is not the case. Copyright holders may still take you to the court. They may, for example, suffer a loss of profit because your use of their material diminishes their ability to sell it: why buy a book when you can read it for free in a corpus (cf. also Amsler 2002)? Copyright issues in corpus creation are complex and unavoidable. While corpus linguists have brought them up periodically for discussion, there is as yet no satisfactory solution to the issue of copyright in corpus creation.

The situation is complicated further by variation in copyright law internationally. According to the copyright law of EU countries, the term of copyright for published works in which the author owns the copyright is the author's lifetime plus 70 years. Under U.S. law, the term of copyright is the author's lifetime plus 50 years; but for works published before 1978, the copyright term is 75 years if the author renewed the copyright after 28 years.

One is able to make some use of copyrighted text without getting clearance, however. Under the convention of "fair dealing" in copyright law, permission need not be sought for short extracts not exceeding 400 words from prose (or a total of 800 words in a series of extracts, none exceeding 300 words); a citation from a poem should not exceed 40 lines or one quarter of the poem. So one can resort to using small samples to build perfectly legal DIY corpora on the grounds of fair usage. But the sizes of such samples are so small as to jeopardize any claim of balance or representativeness.

I maintain that the fair use doctrine as it applies to citations in published works should operate differently when it applies to corpus creation so as to allow corpus creators to build corpora quickly and legally. The limited reproduction of copyrighted works, for instance, in chunks of 3000 words or one-third of the whole text (whichever is shorter) should be protected under fair use for non-profit-making research and educational purposes. A position statement along these lines has been proposed by the corpus using community articulating the point of view that distributing minimal citations of copyrighted texts and allowing the public indirect access to privately held collections of copyrighted texts for statistical purposes are a necessary part of corpus linguistics research and should be inherently protected as fair use, particularly in non-profit-making research contexts (see Cooper 2003). This aim is not a legal reality yet, however. It

will undoubtedly take time for a balance between copyright and fair use for corpus building to develop.

So, what does one do about copyright? My general advice is: if you are in doubt, seek permission. It is usually easier to obtain permission for samples than for full texts, and easier for smaller samples than for larger ones. If you show that you are acting in good faith, and only small samples will be used in non-profit-making research, copyright holders are typically pleased to grant you permission. If some do refuse, you remember it is their right to do so and move on to try other copyright holders until you have enough data.

It appears easier to seek copyright clearance for Web pages on the Internet than for material collected from printed publications. It has been claimed (Spoor 1996: 67) that a vast majority of the documents published on the Internet are not protected by copyright, and that many authors of texts are happy to be able to reach as many people as possible. However, readers should bear in mind that this may not be the case. For example, Cornish (1999: 141) argues that probably all material available on the Web is copyrighted, and that digital publications should be treated the same way as printed works.

Copyright law is generally formulated to prevent someone from making money from selling intellectual property belonging to other people. Unless you are making money using the intellectual property of other people, or you are somehow causing a loss of income to them, it is quite unlikely that copyright problems will arise when building a corpus. Yet copyright law is in its infancy. Different countries have different rules, and it has been argued that with reference to corpora and copyright there is very little which is obviously legal or illegal (cf. Kilgariff 2002). My final word of advice is: proceed with caution.

7.5 Corpus Markup and Annotation

Data collected using a sampling frame as discussed in Section 7.3 forms a raw corpus. Yet such data typically needs to be processed before use. For example, spoken data needs to be transcribed from audio/video recordings; written texts may need to be rendered machine readable, if they are not already, by keyboarding or OCR scanning. Beyond this basic processing, however, lies another form of preparatory work—corpus markup. In addition, in order to extract linguistic information from a corpus, such information must first of all be encoded in the corpus, a process that is technically known as “corpus annotation.”

Corpus markup is a system of standard codes inserted into a document stored in electronic form to provide information about the text itself (i.e., text metadata) and govern formatting, printing or other processing (i.e., structural organization). While metadata markup can be embedded in the same document or stored in a separate but linked document (see below for further discussion of embedding vs. stand-alone annotation), structural markup has to be embedded in the text. Both types of markups are important in corpus creation for at least three reasons. First, the corpus data basically consists of samples of used language. This means that these examples of linguistic usage are taken out of the context in which they originally occurred and their contextual information is lost. Burnard (2002) compares such out-of-context examples to a laboratory specimen and argues that contextual information (i.e., metadata or “data about data”) is needed to restore the context and to enable us to relate the specimen to its original habitat. In corpus creation, therefore, it is important to recover as much contextual information as practically possible to alleviate or compensate for such a loss. Second, while it is possible to group texts and/or transcripts of similar quality together and name these files consistently (e.g., as happens with the LOB and Brown corpora), filenames can provide only a tiny amount of extra-textual information (e.g., text types for written data and sociolinguistic variables of speakers for spoken data) and no textual information (e.g., paragraph/sentence boundaries and speech turns) at all. Yet such data are of great interest to linguists as well as NLP researchers and thus should be encoded, separately from the corpus data *per se*, in a corpus. Markup adds value to a corpus and allows for a broader range of research questions to be addressed as a result. Finally, preprocessing written texts, and particularly transcribing spoken data, also involves markup. For example, in written data, when graphics/tables are removed from the original texts, placeholders must be inserted to indicate the locations and types of omissions; quotations in foreign

languages should also be marked up. In spoken data, pausing and paralinguistic features such as laughter

need to be marked up. Corpus markup is also needed to insert editorial comments, which are sometimes necessary in preprocessing written texts and transcribing spoken data. What is done in corpus markup has a clear parallel in existing linguistic transcription practices. Markup is essential in corpus creation.

Having established that markup is important in corpus creation, we can now move on to discuss markup schemes. It goes without saying that extra-textual and textual information should be kept separate from the corpus data (texts or transcripts) proper. Yet there are different schemes one may use to achieve this goal. One of the earliest markup schemes was COCOA. COCOA references consist of a set of attribute names and values enclosed in angled brackets, as in `<A WILLIAMSHAKESPEARE>`, where A (author) is the attribute name and WILLIAM SHAKESPEARE is the attribute value. COCOA references, however, only encode a limited set of features such as authors, titles, and dates (cf. McEnery and Wilson 2001: 35). Recently, a number of more ambitious metadata markup schemes have been proposed, including for example, the Dublin Core Metadata Initiative (DCMI, see Dekkers and Weibel 2003), the Open Language Archives Community (OLAC, see Bird and Simons 2000), the ISLE Metadata Initiative (IMDI, see Wittenburg et al. 2002), the Text Encoding Initiative (TEI, see Sperberg-McQueen and Burnard 2002), and the Corpus Encoding Standard (CES, see Ide and Priest-Dorman 2000). DCMI provides 15 elements used primarily to describe authored Web resources. OLAC is an extension of DCMI, which introduces refinements to narrow down the semantic scope of DCMI elements and adds an extra element to describe the language(s) covered by the resource. IMDI applies to multimedia corpora (see Section 7.7) and lexical resources as well. From even this brief review it should be clear that there is currently no widely agreed standard way of representing metadata, though all of the current schemes do share many features and similarities. Possibly the most influential schemes in corpus building are TEI and CES, hence I will discuss both of these in some detail here.

The Text Encoding Initiative (TEI) was sponsored by three major academic associations concerned with humanities computing: the Association for Computational Linguistics (ACL), the Association for Literary and Linguistic Computing (ALLC), and the Association for Computers and the Humanities (ACH). The aim of the TEI guidelines is to facilitate data exchange by standardizing the markup or encoding of information stored in electronic form. In TEI, each individual text (referred to as “document”) consists of two parts: header (typically providing text metadata) and body (i.e., the text itself), which are in turn composed of different “elements.” In a TEI header (tagged as `<teiHeader>`), for example, there are four principal elements (see Burnard 2002):

- A file description (tagged as `<fileDesc>`) containing a full bibliographic description of an electronic file.
- An encoding description (tagged as `<encodingDesc>`), which describes the relationship between an electronic text and the source or sources from which it was derived.
- A text profile (tagged as `<profileDesc>`), containing a detailed description of non-bibliographic aspects of a text, specifically the languages and sublanguages used, the situation in which it was produced, the participants and their setting.
- A revision history (tagged as `<revisionDesc>`), which records the changes that have been made to a file.

Each element may contain embedded sub-elements at different levels. Of these, however, only `<fileDesc>` is required to be TEI-compliant; all of the others are optional. Hence, a TEI header can be very complex, or it can be very simple, depending upon the document and the degree of bibliographic control sought. The body part of a TEI document is also conceived as being composed of elements. In this case, an element can be any unit of text, for example, chapter, paragraph, sentence, or word. Formal markup in the body (i.e., structural markup) is by far rarer than in the header (for metadata markup). It is primarily used to encode textual structures such as paragraphs and sentences. Note that the TEI scheme applies to both the markup of metadata and textual structure as well as the annotation of interpretative linguistic analysis.

The TEI scheme can be expressed using a number of different formal languages. The first editions used

the Standard Generalized Markup Language (SGML); the more recent editions (i.e., TEI P4, 2002 and

TEI P5, 2007) can be expressed in the Extensible Markup Language (XML). SGML and XML are very similar, both defining a representation scheme for texts in electronic form, which is device and system independent. SGML is a very powerful markup language, but associated with this power is complexity. XML is a simplified subset of SGML intended to make SGML easy enough for use on the Web. Hence, while all XML documents are valid SGML documents, the reverse is not true. Nevertheless, there are some important surface differences between the two markup languages. End tags can optionally be left out in SGML but they cannot in XML. An attribute name (i.e., generic identifier) in SGML may or may not be case sensitive, but it is always case sensitive in XML. Unless it contains spaces or digits, an attribute value in SGML may be given without double (or single) quotes whereas quotes are mandatory in XML.

As the TEI guidelines are expressly designed to be applicable across a broad range of applications and disciplines, treating not only textual phenomena, they are designed for maximum generality and flexibility (cf. Ide 1998). As such, about 500 elements are predefined in the TEI guidelines. While these elements make TEI very powerful and suitable for the general purpose encoding of electronic texts, they also add complexity to the scheme. In contrast, the Corpus Encoding Standard (CES) is designed specifically for the encoding of language corpora. CES is described as “simplified” TEI in that it includes only the subset of the TEI tagset relevant to corpus-based work. While it simplifies the TEI specifications, CES also extends the TEI guidelines by adding new elements not covered in TEI, specifying the precise values for some attributes, marking required/recommended/optional elements, and explicating detailed semantics for elements relevant to language engineering (e.g., sentence, word, etc.) (cf. Ide 1998).

CES covers three principal types of markups: (1) document-wide markup, which uses more or less the same tags as for TEI to provide a bibliographic description of the document, encoding description, etc.; (2) gross structural markup, which encodes structural units of text (such as volume, chapter, etc.) down to the level of paragraph (but also including footnotes, titles, headings, tables, figures, etc.) and specifies normalization to recommended character sets and entities; (3) markup for sub-paragraph structures, including sentences, quotations, word abbreviations, names, dates, terms and cited words, etc. (see Ide 1998).

CES specifies a minimal encoding level that corpora must achieve to be considered standardized in terms of descriptive representation as well as general architecture. Three levels of text standardization are specified in CES: (1) the metalanguage level, (2) the syntactic level, and (3) the semantic level. Standardization at the metalanguage level regulates the form of the syntactic rules and the basic mechanisms of markup schemes. Users can use a TEI-compliant Document Type Definition (DTD) to define tag names as well as “document models” that specify the relations among tags. As texts may still have different document structures and markups even with the same metalanguage specifications, standardization at the syntactic level specifies precise tag names and syntactic rules for using the tags. It also provides constraints on content. However, the data sender and the data receiver can interpret even the same tag names differently. For example, a <title> element may be intended by the data sender to indicate the name of a book while the data receiver is under no obligation to interpret it as such, because the element can also show a person’s rank, honor, and occupation, etc. This is why standardization at the semantic level is useful. In CES, the <h.title> element only refers to the name of a document. CES seeks to standardize at the semantic level for those elements most relevant to language engineering applications, in particular, linguistic elements. The three levels of standardization are designed to achieve the goal of universal document interchange. Like the TEI scheme, CES not only applies to corpus markup, it also covers encoding conventions for the linguistic annotation of text and speech, currently including morpho-syntactic tagging (i.e., part-of-speech tagging, see Chapter 10) and parallel text alignment in parallel corpora (see Chapter 16).

CES was developed and recommended by the Expert Advisory Groups on Language Engineering Standards (EAGLES) as a TEI-compliant application of SGML that could serve as a widely accepted set of encoding standards for corpus-based work. CES is available in both SGML and XML versions. The XML version, referred to as XCES, has also developed support for additional types of annotation and resources, including discourse/dialogue, lexicons, and speech (Ide et al. 2000). On the other hand, while metalanguages such as SGML and XML usually follow the system of attribute names laid out in

implementation standards such as TEI and CES, this may not be necessarily the case.

Closely related to corpus markup is annotation, but the two are different. As annotation is so important in corpus creation and NLP research that specific types of annotation merit in-depth discussions in separate chapters (e.g., Chapters 8, 10, and 14), here I will only discuss annotation briefly. Corpus annotation can be defined as the process of “adding such interpretative, linguistic information to an electronic corpus of spoken and/or written language data” (Leech 1997: 2). While annotation defined in a broad sense may refer to the encoding of both textual/contextual information and interpretative linguistic analysis, as shown by the conflation of the two often found in the literature, the term is used in a narrow sense here, referring solely to the encoding of linguistic analyses such as part-of-speech tagging and syntactic parsing in a corpus text.

Corpus annotation, as used in a narrow sense, is fundamentally distinct from markup, though the distinction is not accepted by all and the two terms are sometimes used interchangeably in the literature. Corpus markup provides relatively objectively verifiable information regarding the components of a corpus and the textual structure of each text. In contrast, corpus annotation is concerned with interpretative linguistic information. “By calling annotation ‘interpretative,’ we signal that annotation is, at least in some degree, the product of the human mind’s understanding of the text” (Leech 1997: 2). For example, the part of speech of a word may be ambiguous and hence is more readily defined as corpus annotation than corpus markup. On the other hand, the sex of a speaker or writer is normally objectively verifiable and as such is a matter of markup, not annotation.

Corpus annotation can be undertaken at different levels and may take various forms. For example, at the phonological level, corpora can be annotated for syllable boundaries (phonetic/phonemic annotation) or prosodic features (prosodic annotation); at the morphological level corpora can be annotated in terms of prefixes, suffixes and stems (morphological annotation); at the lexical level, corpora can be annotated for parts-of-speech (POS tagging), lemmas (lemmatization), and semantic fields (semantic annotation); at the syntactic level, corpora can be annotated with syntactic analysis (parsing, treebanking, or bracketing); at the discursial level, corpora can be annotated to show anaphoric relations (coreference annotation), pragmatic information like speech acts (pragmatic annotation) or stylistic features such as speech and thought presentation (stylistic annotation). Of these the most widespread type of annotation is part-of-speech tagging (see Chapter 10), which has been successfully applied to many languages; syntactic parsing is also developing rapidly (see Chapters 8 and 11) while some types of annotation (e.g., discursial and pragmatic annotations) are presently relatively undeveloped.

I have so far assumed that the process of annotation leads to information being mixed in the original corpus text or so-called base document when it is applied to a corpus (i.e., the annotation becomes so-called embedded annotation). However, the Corpus Encoding Standard recommends the use of “stand-alone annotation,” whereby the annotation information is retained in separate SGML/XML documents (with different Document Type Definitions) and linked to the original and other annotation documents in hypertext format. In contrast to embedded annotation, stand-alone annotation has a number of advantages (Ide 1998):

- It provides control over the distribution of base documents for legal purposes.
- It enables annotation to be performed on base documents that cannot easily be altered (e.g., they are read-only).
- It avoids the creation of potentially unwieldy documents.
- It allows multiple overlapping hierarchies.
- It allows for alternative annotation schemes to be applied to the same data (e.g., different POS tagsets).
- It enables new annotation levels to be added without causing problems for existing levels of annotation or search tools.
- It allows annotation at one level to be changed without affecting other levels.

Stand-alone annotation is in principle ideal and is certainly technically feasible (see Thompson and

McKelvie 1997). It may also represent the future standard for certain types of annotation. In addition,

the stand-alone architecture can facilitate multilevel or multilayer annotations as well (see Dipper 2005). Presently, however, there are two problems associated with stand-alone annotation. The first issue is related to the complexity of corpus annotation. As noted earlier, annotation may have multiple forms in a corpus. While some of these readily allow for the separation of annotation codes from base documents (e.g., lemmatization, part-of-speech tagging, and semantic annotation), others may involve much more complexity in establishing links between codes and annotated items (e.g., coreference and stylistic annotations). Even if such links can be established, they are usually prone to error. The second issue is purely practical. As far as I am aware, the currently available corpus exploration tools, including the latest versions of WordSmith (versions 4.0 and 5.0) and Xaira (Burnard and Todd 2003), have all been designed for use with embedded annotation. Stand-alone annotation, while appealing, is only useful when appropriate search tools are available for use on stand-alone annotated corpora.

7.6 Multilingual Corpora

I have so far assumed in this chapter that a corpus only involves one language. Corpora of this kind are monolingual. But there are also corpora that cover more than one language, which are referred to as multilingual corpora. In this section, I will shift my focus to the multilingual dimension of corpus creation.

With ever increasing international exchange and accelerated globalization, translation and contrastive studies are more popular than ever. As part of this new wave of research on translation and contrastive studies, multilingual corpora such as parallel and comparable corpora are playing an increasingly prominent role. As Aijmer and Altenberg (1996: 12) observe, parallel and comparable corpora “offer specific uses and possibilities” for contrastive and translation studies:

- They give new insights into the languages compared – insights that are not likely to be gained from the study of monolingual corpora.
- They can be used for a range of comparative purposes and increase our knowledge of language-specific, typological and cultural differences, as well as of universal features.
- They illuminate differences between source texts and translations, and between native and nonnative texts.
- They can be used for a number of practical applications, for example, in lexicography, language teaching, and translation.

In addition to these benefits of multilingual resources in linguistic research, we can also add to the list the fact that aligned parallel corpora are indispensable to the development of NLP applications such as computer-aided translation and machine translation (see Chapters 17 and 18) and multilingual information retrieval and extraction (see Chapters 19 and 21).

A multilingual corpus involves texts of more than one language. As corpora that cover two languages are conventionally known as “bilingual,” multilingual corpora, in a narrow sense, must involve more than two languages, though “multilingual” and “bilingual” are often used interchangeably in the literature, and also in this chapter. A multilingual corpus can be a parallel corpus, or a comparable corpus. Given that corpora involving more than one language are a relatively new phenomenon, with most related research hailing from the early 1990s, it is unsurprising to discover that there is some confusion surrounding the terminology used in relation to these corpora.

It can be said that terminological confusion in multilingual corpora centers on two terms: “parallel” and “comparable.” For some scholars (e.g., Aijmer and Altenberg 1996; Granger 1996: 38), corpora composed of source texts in one language and their translations in another language (or other languages) are “translation corpora” while those comprising different components sampled from different native languages using comparable sampling techniques are called “parallel corpora.” For others (e.g., Baker 1993: 248, 1995, 1999; Barlow 1995, 2000: 110; Hunston 2002: 15; McEnery and Wilson 1996: 57; McEnery

et al. 2006), corpora of the first type are labeled “parallel” while those of the latter type are comparable corpora. As argued in McEnery and Xiao (2007a: 19–20), while different criteria can be used to define different types of corpora, they must be used consistently and logically. For example, we can say that a corpus is monolingual, bilingual, or multilingual if we take the number of languages involved as the criterion for definition. We can also say that a corpus is a translation or a non-translation corpus if the criterion of corpus content is used. But if we choose to define corpus types by the criterion of corpus form, we must use the terminology consistently. Then we can say a corpus is parallel if the corpus contains source texts and translations in parallel, or it is a comparable corpus if its components or subcorpora are comparable by applying the same sampling frame. It is illogical, however, to refer to corpora of the first type as translation corpora by the criterion of content while referring to corpora of the latter type as comparable corpora by the criterion of form.

Additionally, a parallel corpus, in my terms, can be either unidirectional (e.g., from English into Chinese or from Chinese into English alone), or bidirectional (e.g., containing both English source texts with their Chinese translations as well as Chinese source texts with their English translations), or multidirectional (e.g., the same piece of text with its Chinese, English, French, Russian, and Arabic versions). In this sense, texts that are produced simultaneously in different languages (e.g., UN regulations) also belong to the category of parallel corpora. A parallel corpus must be aligned at a certain level (for instances, at document, paragraph, sentence, or word level) in order to be useful. The automatic alignment of parallel corpora is not a trivial task for some language pairs, though alignment is generally very reliable for many closely related European language pairs (cf. McEnery et al. 2006: 50–51; see Chapter 16 for further discussion).

Another complication in terminology involves a corpus that is composed of different variants of the same language. This is particularly relevant to translation studies because it is a very common practice in this research area to compare a corpus of translated texts—that I call a “translational corpus”—and a corpus consisting of comparably sampled non-translated texts in the same language (see Xiao and Yue 2009). They form a monolingual comparable corpus. To us, a multilingual comparable corpus samples different native languages, with its comparability lying in the matching or comparable sampling techniques, similar balance (i.e., coverage of genres and domains) and representativeness, and similar sampling period (see Section 7.3). By my definition, corpora containing different regional varieties of the same language (e.g., the International Corpus of English, ICE) are not comparable corpora because all corpora, as a resource for linguistic research, have “always been pre-eminently suited for comparative studies” (Aarts 1998: ix), either intralingually or interlingually. The Brown, LOB, Frown, and FLOB corpora are also used typically for comparing language varieties synchronically and diachronically. Corpora such as these can be labeled as “comparative corpora.” They are not “comparable corpora” as suggested in the literature (e.g., Hunston 2002: 15).

Having clarified some terminological confusion in multilingual corpus research, it is worth pointing out the distinctions discussed here are purely for the sake of clarification. In reality, there are multilingual corpora that are a mixture of parallel and comparable corpora. For example, in spite of its name, the English–Norwegian Parallel Corpus (ENPC) can be considered as a combination of a parallel and comparable corpus. I will not discuss the state of the art of multilingual corpus research here. Interested readers are advised to refer to McEnery and Xiao (2007b).

Multilingual corpora often involve a writing system that relies heavily on non-ASCII characters. Character encoding is rarely an issue in corpus creation for alphabetical languages (e.g., English) that use ASCII characters. However, even languages that use a small number of accented Latin characters may have encountered encoding problems. For monolingual corpora of many other languages that use different writing systems, especially for multilingual corpora that contain a wide range of writing systems, encoding is all the more important if one wants to display the corpus properly or facilitate data interchange. For example, Chinese can be encoded using GB2312 (Simplified Chinese), Big5 (Traditional Chinese), or Unicode (UTF-8, UTF-7 or UTF-16). Both GB2312 and Big5 are 2-byte encoding systems that require

language-specific operating systems or language-support packs if the Chinese characters encoded are to be

displayed properly. Language specific encoding systems such as these make data interchange problematic. It is also quite impossible to display a document containing both simplified and traditional Chinese characters using these encoding systems. As McEnery et al. (2000) note, the main difficulty in building a multilingual corpus of Asian languages is the need to standardize the language data into a single character set. Unicode is recommended as a solution to this problem (see McEnery and Xiao 2005). Unicode is truly multilingual in that it can display characters from a very large number of writing systems. From the Unicode Standard version 1.1 onward, Unicode is fully compatible with ISO 10646-1 (UCS). The combination of Unicode and XML is a general trend in corpus creation (see Xiao et al. 2004). As such, it is to be welcomed.

7.7 Multimodal Corpora

The corpora discussed so far in this chapter, whether spoken or written, have been assumed to be text-based; that is, spoken language is treated as if it is written. In this text-based approach to corpus creation, audio/video recordings of spoken data are transcribed, with the transcript possibly also including varying levels of details of spoken features (e.g., turn overlaps) and paralinguistic features (e.g., laughter). Corpus analysis is then usually undertaken on the textual transcript without reference to the original recording unless one is engaged in prosodic or phonetic research.

As noted in Section 7.5, a corpus is essentially a collection of samples of used language, which have been likened to a laboratory specimen out of its original habitat (Burnard 2005). While corpus markup can help to restore some contextual information, a large part of such information is lost, especially in transcripts of video clips. As Kress and van Leeuwen (2006: 41) observe, “a spoken text is never just verbal, but also visual combining with modes such as facial expressions, gesture, posture and other forms of self-presentation,” the latter of which cannot be captured and transcribed easily, if at all. Consequently, “even the most detailed, faithful and sympathetic transcription cannot hope to capture” spoken language (Carter 2004: 26). As such, there has recently been an increasing interest in multimodal corpora. In this kind of corpora, annotated transcripts are aligned with digital audio/video clips with the help of time stamps, which not only renders the corpus searchable with the help of transcripts but also allows the user to access the segments of recordings corresponding to the search results. There are a number of existing multimodal corpora including, for example, the Nottingham Multi-Modal Corpus (NMMC, see Adolphs and Carter 2007), the Singapore Corpus of Research in Education (SCoRE, see Hong 2005), Padova Multimedia English Corpus (see Ackerley and Coccetta 2007), and the Spoken Chinese Corpus of Situated Discourse (SCCSD, see Gu 2002).

Multimodal corpora and multimodal concordancers are still in their infancy (Baldry 2006: 188). They are technically more challenging to develop than purely text-based corpora and corpus tools. However, given the special values of such corpora, and the advances of technologies (e.g., those that help to track and annotate gestures), multimodal corpora will become more common and more widely used in the near future.

7.8 Conclusions

This chapter has focused on corpus creation, covering the major factors that must be taken into account in this process. I have discussed both core issues relating to corpus design (e.g., corpus size, representativeness, and balance) as well as corpus processing (e.g., data collection, markup, and annotation), and peripheral issues such as multilingual and multimodal corpora.

One important reason for using corpora is to extract linguistic information present in those corpora. But it is often the case that in order to extract such information from a corpus, a linguistic analysis must

first be encoded in the corpus. Such annotation adds value to a corpus in that it considerably extends the

range of research questions that a corpus can readily address. In this chapter, I have discussed corpus annotation in very general terms. The chapter that follows will explore annotation in greater depth.

References

- Aarts, J. (1998) Introduction. In S. Johansson and S. Oksefjell (eds.), *Corpora and Cross-Linguistic Research*, pp. ix–xiv. Amsterdam, the Netherlands: Rodopi.
- Ackerley, K. and Cocchetta, F. (2007) Enriching language learning through a multimedia corpus. *ReCALL* 19(3): 351–370.
- Adolphs, S. and Carter, R. (2007) Beyond the word: New challenges in analyzing corpora of spoken English. *European Journal of English Studies* 11(2): 133–146.
- Aijmer, K. and Altenberg, B. (1996) Introduction. In K. Aijmer, B. Altenberg and M. Johansson (eds.), *Language in contrast. Papers from Symposium on Text-Based Cross-Linguistic Studies, Lund, Sweden, March 1994*, pp. 10–16. Lund, Sweden: Lund University Press.
- Amsler, R. (2002) Legal aspects of corpora compiling. In *Corpora List Archive on 1st October 2002*. URL: <http://helmer.hit.uib.no/corpora/2002-3/0256.html>.
- Aston, G. and Burnard, L. (1998) *The BNC Handbook*. Edinburgh, U.K.: Edinburgh University Press.
- Atkins, S., Clear, J., and Ostler, N. (1992) Corpus design criteria. *Literary and Linguistic Computing* 7(1): 1–16.
- Baker, M. (1993) Corpus linguistics and translation studies: Implications and applications. In M. Baker, G. Francis, and E. Tognini-Bonelli (eds.), *Text and Technology: In Honour of John Sinclair*, pp. 233–352. Amsterdam, the Netherlands: Benjamins.
- Baker, M. (1995) Corpora in translation studies: An overview and some suggestions for future research. *Target* 7: 223–243.
- Baker, M. (1999) The role of corpora in investigating the linguistic behaviour of professional translators. *International Journal of Corpus Linguistics* 4: 281–298.
- Baldry, A. P. (2006) The role of multimodal concordancers in multimodal corpus linguistics. In T. D. Royce and W. L. Bowcher (eds.), *New Directions in the Analysis of Multimodal Discourse*, pp. 173–214. London, U.K.: Routledge.
- Barlow, M. (1995) *A Guide to ParaConc*. Huston, TX: Athelstan.
- Barlow, M. (2000) Parallel texts and language teaching. In S. Botley, A. McEnery, and A. Wilson (eds.), *Multilingual Corpora in Teaching and Research*, pp. 106–115. Amsterdam, the Netherlands: Rodopi.
- Baroni, M. (2009) Distributions in text. In A. Lüdeling and M. Kytö (eds.), *Corpus Linguistics: An International Handbook* (Vol. 2), pp. 803–822. Berlin, Germany: Mouton de Gruyter.
- Baroni, M. and Bernardini, S. (2004) BootCaT: Bootstrapping corpora and terms from the Web. In M. Lino, M. Xavier, F. Ferreira, R. Costa, and R. Silva (eds.), *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC) 2004*, Lisbon, Portugal, May 24–30, 2004. Baroni, M. and Bernardini, S. (eds.). (2006) *Wacky! Working Papers on the Web as Corpus*. Bologna, Italy: GEDIT.
- Belica, C. (1996) Analysis of temporal change in corpora. *International Journal of Corpus Linguistics*

1(1): 61–74.

Biber, D. (1988) *Variation Across Speech and Writing*. Cambridge, U.K.: Cambridge University Press. Biber, D. (1993) Representativeness in corpus design. *Literary and Linguistic Computing* 8(4): 243–257. Bird, S. and Simons, G. (2000) *White Paper on Establishing an Infrastructure for Open Language Archiving*.

URL: <http://www.language-archives.org/docs/white-paper.html>.

Burnard, L. (2002) *Validation Manual for Written Language Resources*. URL: <http://www.oucs.ox.ac.uk/rt/elra/D1.xml>.

Burnard, L. (2005) Metadata for corpus work. In M. Wynne (ed.), *Developing Linguistic Corpora: A Guide to Good Practice*, pp. 30–46. Oxford, U.K.: AHDS.

- Burnard, L. and Todd, T. (2003) Xara: An XML aware tool for corpus searching. In D. Archer, P. Rayson, A. Wilson, and A. McEnery (eds.), *Proceedings of Corpus Linguistics 2003*, Lancaster, U.K., pp. 142–144. Lancaster, U.K.: Lancaster University.
- Carter, R. (2004) Grammar and spoken English. In C. Coffin, A. Hewings, and K. O'Halloran (eds.), *Applying English Grammar: Corpus and Functional Approaches*, pp. 25–39. London, U.K.: Arnold.
- Collins (2007) *Collins English Dictionary* (9th ed.). Toronto, Canada: HarperCollins.
- Cooper, D. (2003) Legal aspects of corpora compiling. In *Corpora List Archive on 19th June 2003*. URL: <http://helmer.aksis.uib.no/corpora/2003-1/0596.html>.
- Cornish, G. P. (1999) *Copyright: Interpreting the Law for Libraries, Archives and Information Services* (3rd ed.). London, U.K.: Library Association Publishing.
- Dekkers, M. and Weibel, S. (2003) State of the Dublin core metadata initiative. *D-Lib Magazine* 9(4). URL: <http://www.dlib.org/dlib/april03/weibel/04weibel.html>.
- Dipper, S. (2005) XML-based stand-off representation and exploitation of multi-level linguistic annotation. In *Proceedings of Berliner XML Tage 2005 (BXML 2005)*, Berlin, Germany, pp. 39–50.
- Dipper, S. (2008) Theory-driven and corpus-driven computational linguistics, and the use of corpora. In A. Ludeling and M. Kyto (eds.), *Corpus Linguistics: An International Handbook* (Vol. 1), pp. 68–96. Berlin, Germany: Mouton de Gruyter.
- Evert, S. (2006) How random is a corpus? The library metaphor. *Zeitschrift für Anglistik und Amerikanistik* 54(2): 177–190.
- Givón, T. (1995) *Functionalism and Grammar*. Amsterdam, the Netherlands: John Benjamins.
- Granath, S. (2007) Size matters — Or thus can meaningful structures be revealed in large corpora. In R. Facchinetti (ed.), *Corpus Linguistics 25 Years On*, pp. 169–185. Amsterdam, the Netherlands: Rodopi.
- Granger, S. (1996) From CA to CIA and back: An integrated approach to computerized bilingual and learner corpora. In K. Aijmer, B. Altenberg, and M. Johansson (eds.), *Language in contrast. Symposium on Text-based Cross-linguistic Studies, Lund, Sweden, March 1994*, pp. 38–51. Lund, Sweden: Lund University Press.
- Gu, Y. (2002) Towards an understanding of workplace discourse. In C. N. Candlin (ed.), *Research and Practice in Professional Discourse*, pp. 137–86. Hong Kong: City University of Hong Kong Press.
- Hakulinen, A., Karlsson, F., and Vilkkumäki, M. (1980) *Suomen tekstilauseiden piirteitä: kvantitatiivinen tutkimus*. Department of General Linguistics, University of Helsinki, Helsinki, Finland, Publications No. 6.
- Hausser, H. (1999) *Functions of Computational Linguistics*. Berlin, Germany: Springer-Verlag.
- Hong, H. (2005) SCORE: A multimodal corpus database of education discourse in Singapore schools. In *Proceedings of Corpus Linguistics 2005*. <http://www.corpus.bham.ac.uk/plc/ScopeHong.pdf> Hundt, M., Sand, A., and Siemund, R. (1998) *Manual of Information to Accompany the Freiburg-LOB Corpus of British English ('FLOB')*. URL: <http://khnt.hit.uib.no/icame/manuals/flob/INDEX.HTM>. Hundt, M., Sand, A., and Skandera, P. (1999) *Manual of Information to Accompany the Freiburg-Brown Corpus of American English ('Frown')*. URL: <http://khnt.hit.uib.no/icame/manuals/frown/INDEX.HTM>.

- Hundt, M., Biewer, C., and Nesselhauf, N. (eds.). (2007) *Corpus Linguistics and the Web*. Amsterdam, the Netherlands: Rodopi.
- Hunston, S. (2002) *Corpora in Applied Linguistics*. Cambridge, U.K.: Cambridge University Press.
- Ide, N. (1998) Corpus encoding standard: SGML guidelines for encoding linguistic corpora. In *LREC-1998 Proceedings*, Granada, Spain, pp. 463–470.
- Ide, N. and Priest-Dorman, G. (2000) *Corpus Encoding Standard—Document CES1*. URL: <http://www.cs.vassar.edu/CES/>.
- Ide, N., Patrice, B., and Romary L. (2000) XCES: An XML-based encoding standard for linguistic corpora. In *LREC-2000 Proceedings*, Athens, Greece, pp. 825–830.

- Johansson, S., Leech, G., and Goodluck, H. (1978) *Manual of Information to Accompany the Lancaster-Oslo/Bergen Corpus of British English, for Use with Digital Computers*. Oslo, Norway: University of Oslo.
- Keller, F. and Lapata, M. (2003) Using the Web to obtain frequencies for unseen bigrams. *Computational Linguistics* 29(3): 459–484.
- Kilgarrriff, A. (2002) Legal aspects of corpora compiling. In *Corpora List Archive on 1st October 2002*.
URL: <http://helmer.hit.uib.no/corpora/2002-3/0253.html>.
- Kilgarrriff, A. and Grefenstette, G. (eds.). (2003) Special Issue on Web as Corpus. *Computational Linguistics* 29(3): 333–502.
- Kress, G. and van Leeuwen, T. (2006) *Reading Images: The Grammar of Visual Design* (2nd ed.). London, U.K.: Routledge.
- Krishnamurthy, R. (2000) Size matters: Creating dictionaries from the world's largest corpus. In *Proceedings of KOTESOL 2000: Casting the Net: Diversity in Language Learning*, Taegu, Korea, pp. 169–180.
- Kučera, H. and Francis, W. (1967) *Computational Analysis of Present-Day English*. Providence, RI: Brown University Press.
- Leech, G. (1991) The state of art in corpus linguistics. In K. Aijmer and B. Altenberg (eds.), *English Corpus Linguistics*, pp. 8–29. London, U.K.: Longman.
- Leech, G. (1997) Introducing corpus annotation. In R. Garside, G. Leech, and A. McEnery (eds.), *Corpus Annotation*, pp. 1–18. London, U.K.: Longman.
- Manning, C. and Schütze, H. (1999) *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.
- McEnery, A. and Wilson, A. (1996/2001) *Corpus Linguistics* (2nd ed. 2001). Edinburgh, U.K.: Edinburgh University Press.
- McEnery, A. and Xiao, R. (2005) Character encoding in corpus construction. In M. Wynne (ed.), *Developing Linguistic Corpora: A Guide to Good Practice*, pp. 47–58. Oxford, U.K.: AHDS.
- McEnery, A. and Xiao, R. (2007a) Parallel and comparable corpora: What is happening? In M. Rogers and G. Anderman (eds.), *Incorporating Corpora: The Linguist and the Translator*, pp. 18–31. Clevedon, U.K.: Multilingual Matters.
- McEnery, A. and Xiao, R. (2007b) Parallel and comparable corpora: The state of play. In Y. Kawaguchi, T. Takagaki, N. Tomimori, and Y. Tsuruga (eds.), *Corpus-Based Perspectives in Linguistics*, pp. 131–145. Amsterdam, the Netherlands: John Benjamins.
- McEnery, A., Baker, P., Gaizauskas, R., and Cunningham, H. (2000) EMILLE: Building a corpus of South Asian languages. *Vivek: A Quarterly in Artificial Intelligence* 13(3): 23–32.
- McEnery, A., Baker, P., and Cheepen, C. (2001) Lexis, indirectness and politeness in operator calls. In C. Meyer and P. Leistyna (eds.), *Corpus Analysis: Language Structure and Language Use*. Amsterdam, the Netherlands: Rodopi.
- McEnery, A., Xiao, R., and Tono, Y. (2006) *Corpus-Based Language Studies: An Advanced Resource Book*. London, U.K.: Routledge.
- Ostler, N. (2008) Corpora of less studied languages. In A. Ludeling and M. Kyto (eds.), *Corpus Linguistics: An International Handbook* (Vol. 1), pp. 457–484. Berlin, Germany: Mouton de Gruyter.

- Otlogetswe, T. (2004) The BNC design as a model for a Setswana language corpus. In *Proceeding of the Seventh Annual CLUK Research Colloquium*, pp. 93–198. University of Birmingham, Edgbaston, U.K., January 6–7, 2004.
- Piao, S., Wilson, A., and McEnery, A. (2002) A multilingual corpus toolkit. *Paper Presented at the Fourth North American Symposium on Corpus Linguistics*, Indianapolis, IN, November 1–3, 2002.
- Santos, D. (1996) Tense and aspect in English and Portuguese: A contrastive semantical study. PhD thesis, Universidade Tecnica de Lisboa, Lisbon, Portugal.
- Scott, M. (2003) *WordSmith Tools Manual*. URL: <http://www.lexically.net/wordsmith/version4/>.

- Shimazumi, M. and Berber-Sardinha, A. (1996) Approaching the assessment of performance unit (APU) archive of schoolchildren's writing from the point of view of corpus linguistics. *Paper Presented at the TALC'96 Conference*, Lancaster University, Lancaster, U.K., August 11, 1996.
- Sinclair, J. (1991) *Corpus Concordance Collocation*. Oxford, U.K.: Oxford University Press.
- Sinclair, J. (2004) *Trust the Text: Language, Corpus and Discourse*. London, U.K.: Routledge.
- Sinclair, J. (2005) Corpus and Text: Basic Principles. In M. Wynne (ed.), *Developing Linguistic Corpora: A Guide to Good Practice*, pp. 1–20. Oxford, UK: AHDS.
- Sperberg-McQueen, C. M. and Burnard, L. (eds.). (2002) *TEI P4: Guidelines for Electronic Text Encoding and Interchange (XML Version)*. Oxford, U.K.: Text Encoding Initiative Consortium.
- Spoor, J. (1996) The copyright approach to copying on the Internet: (Over)stretching the reproduction right? In H. Hugenholtz (ed.), *The Future of Copyright in a Digital Environment*, pp. 67–80. Dordrecht, the Netherlands: Kluwer Law International.
- Teubert, W. (2000) Corpus linguistics—A partisan view. *International Journal of Corpus Linguistics* 4(1):1–16.
- Thompson, H. and McKelvie, D. (1997) Hyperlink semantics for standoff markup of read-only documents. In *Proceedings of SGML Europe'97*, Barcelona, Spain, May 1997. URL: <http://www.ltg.ed.ac.uk/~ht/sgmleu97.html>.
- Váradi, T. (2000) Corpus linguistics—linguistics or language engineering? In T. Erjavec and J. Gross (eds.), *Information Society Multi-Conference Proceedings Language Technologies*, pp. 1–5. Ljubljana, Slovenia, October 17–18, 2000.
- Wittenburg, P., Peters, W., and Broeder, D. (2002) Metadata proposals for corpora and lexica. In *LREC-2002 Proceedings*, Las Palmas, Spain, pp. 1321–1326.
- Xiao, R. (2008) Well-known and influential corpora. In A. Lüdeling and M. Kyto (eds.), *Corpus Linguistics: An International Handbook* (Vol. 1), pp. 383–457. Berlin, Germany: Mouton de Gruyter.
- Xiao, R. and Yue, M. (2009) Using corpora in translation studies: The state of the art. In P. Baker (ed.), *Contemporary Approaches to Corpus Linguistics*, pp. 237–262. London, U.K.: Continuum.
- Xiao, R., McEnery, A., Baker, P., and Hardie, A. (2004) Developing Asian language corpora: Standards and practice. In *Proceedings of the Fourth Workshop on Asian Language Resources*, Sanya, Hainan Island, pp. 1–8, March 25, 2004.
- Yates, F. (1965) *Sampling Methods for Censuses and Surveys* (3rd ed.). London, U.K.: Charles Griffin and Company Limited.

8

Treebank Annotation

Eva Hajičová
Charles University

Anne Abeillé
Université Paris 7 and CNRS

Jan Hajič
Charles University

Jirí Mírovský
Charles University

Zdeňka Urešová
Charles University

8.1	Introduction	167
8.2	Corpus Annotation Types	168
8.3	Morphosyntactic Annotation	169
8.4	Treebanks: Syntactic, Semantic, and Discourse Annotation	169
	Motivation and Definition · An Example: The Penn Treebank ·	
	Annotation and Linguistic Theory · Going Beyond the Surface Shape of the	
	Sentence	
8.5	The Process of Building Treebanks	176
8.6	Applications of Treebanks	177
8.7	Searching Treebanks	180
8.8	Conclusions	181
	Acknowledgments	182
	References	182

8.1 Introduction

Corpus annotation, whether lexical, morphological, syntactic, semantic, or any other, brings additional linguistic information as an added value to a corpus. The annotation scenario might differ considerably among corpora, but it is always based on some formalism that represents the desired level and area of linguistic interpretation of the corpus. From the simple annotation of part-of-speech categories to the shallow syntactic annotation to semantic role labeling to the “deep,” complex annotation of semantic and discourse relations, there is usually some more or less sound linguistic theory behind the design of the representation used, or at least certain principles common to several such theories.

Corpora have become popular resources for computationally minded linguists and computer science experts developing applications in Natural Language Processing (NLP). Linguists typically look for various occurrences of specific words or patterns to find examples or counterexamples within the theories they build or work with, lexicographers use corpora for creating dictionary entries by looking for evidence of use of words in various senses and contexts, computational linguists together with computer scientists and statisticians construct language models and build part-of-speech taggers, syntactic parsers and various semantic labelers to be used in applications, such as machine translation, information retrieval, information extraction, question answering and summarization systems, dialogue systems and many more. Often, annotated corpora were built by linguists who wanted to confront their theory with real-world texts.

Most of the work on annotated corpora concerns the domain of written texts, on which this chapter is focused. However, it should be acknowledged that the growing interest in the speech community to develop advanced models of spoken language has led to an increasing effort to process corpora that represent the spoken form of language. This is well documented among other things by the contributions in the special issue of the journal *Speech Communication* published in 2001 (Bird and Harrington 2001), in

which topics such as annotation schemes, tools for creating, searching, and manipulating the annotations as well as future directions are discussed. Also, there is an increasing number of contributions at the most recent Interspeech and other speech-related conferences that tackle the topical issues of annotation of spoken language, mostly of dialogues (see e.g., already the Tübingen corpus of spontaneous Japanese dialogues reported on in Kawata and Barteles 2000). Prosody, disfluencies, and dialogue acts (such as statements, requests, etc.) have to be annotated, too.

In this chapter, after introducing types of corpus annotation and briefly mentioning unstructured morphosyntactically annotated corpora, we will focus on treebank annotation—the definition of treebanks, their properties, examples of existing treebanks, their relation to linguistic theory, and the process of annotation and quality control. We will also refer the application of treebanks, and specifically discuss searching for linguistic information in them.

8.2 Corpus Annotation Types

Annotation consists of pieces of information added to the language data. The data may have various forms—it can be audio, video, or textual data. The added information can be external, such as the author's name, the date of recording/writing, or the type of font—this type of annotation is often called “metadata.” We are more interested in linguistic information, such as part of speech, clause boundary, word sense, syntactic analysis, co-reference annotation, etc.

Usually, three linguistic annotation phases (layers, or simply types) are distinguished: a morphosyntactic layer, only dealing with morphosyntactic ambiguity (and part of speech and inflectional and morphological annotation), a layer dealing with syntactic relations of different degree of depth (oriented toward constituency or dependency annotations), and a layer focused on different aspects of semantic and discourse relations such as word sense disambiguation, anaphoric relations, etc.

A morphosyntactic tag includes (part of) the following information (e.g., for the French token *la*):

- Lemma (*le*)
- Part of speech (determiner)
- Subcategory (definite)
- Inflection (feminine singular)

A syntactic tag includes (part of) the following information:

- Constituent boundaries (clause, NP, ...)
- Grammatical function of words or constituents (Subject, Complement, Auxiliary, ...)
- Dependencies between words or constituents (head/dependent)

A semantic tag includes, among other things, (part of) the following information:

- Word sense for ambiguous words (interest-1 (general), interest-2 (banking), ...)
- Domain (or hyperonym)
- Pointer to the antecedent for pronouns or anaphoric elements

A discourse tag includes (part of) the following information:

- Discourse relation (cause, purpose, etc.)
- Temporal relation (simultaneity, anteriority, ...)
- Discourse act (statement, request, ...)

For semantic and discourse annotations, the tagset can be open (if the expected value is a segment of the text) and the annotated element does not have to be a word but a sequence of words or a whole sentence (or utterance) (cf. Craggs and Wood 2005).

A scheme of the annotation types is given in Figure 8.1.

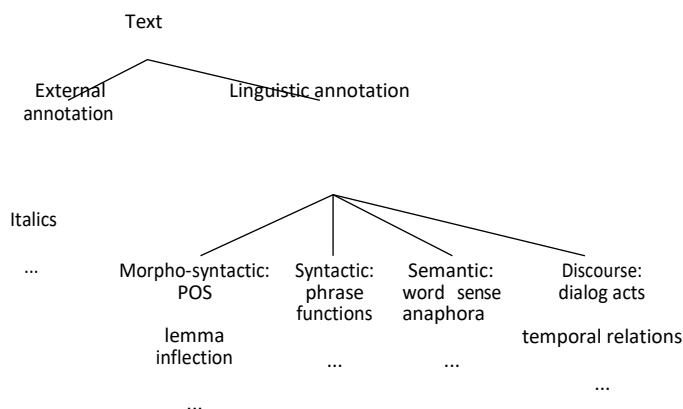


FIGURE 8.1 A scheme of annotation types (layers).

8.3 Morphosyntactic Annotation

The first morphosyntactically tagged corpora were the Brown Corpus for English (Kucera and Francis 1967) and the Lancaster-Oslo-Bergen (LOB) corpus (Johansson 1980). Corpora tagged with morphosyntactic annotation are now available for various languages: the British National Corpus for British English, the Penn Treebank (Marcus et al. 1993) for American English, PAROLE and MULTTEXT corpora for various European languages (Véronis and Khouri 1995), and many others.

Morphosyntactic tagsets (the list of symbols used for categories and inflections) can vary from a dozen (Marcus et al. 1993) to several million (Ofazer et al. 2003), with diverse possibilities in between (see e.g., Hajič and Hladká 1998, Böhmová et al. 2003, Brants et al. 2003). They partly reflect the richness of the language's morphosyntax (with case or agreement systems yielding numerous features to be encoded).

In morphosyntactic annotation, information concerning the presence of so-called named entities (names of persons, organizations, locations, etc.) can be included. For example, proper names are distinguished from common names by special tags in the Penn Treebank and in the Negra Treebank, and by special lemma suffixes in the Prague Dependency Treebank (PDT). However, the annotation of named entities can also be represented independently of any morphosyntactic annotation, as it is the case with popular MUC-6 data (Message Understanding Conference focused on the named entity resolution task, Grishman and Sundheim 1996) or in Ševčíková et al. (2007), in which much richer classification of named entity types are offered (compared to what is usually stored in morphosyntactic tags).

Problematic cases are often the same across languages. Not all morphosyntactic taggers deal with compounds or idioms as such. Some sequences are ambiguous between a compound and a noncompound interpretation (e.g., *sur ce* in French can either be the compound adverb (= *at once*), or a preposition (= *on*) followed by a determiner (= *this*)). Most taggers usually prefer the compound interpretation. Unknown (or misspelled) words, foreign words, and punctuation marks can be ignored or annotated with specific tags. Proper name is often a default tag for all unknown words.

8.4 Treebanks: Syntactic, Semantic, and Discourse Annotation

8.4.1 Motivation and Definition

For simple linguistic queries, morphosyntactically annotated texts described in the previous section enable

a reduction of the “noise” usually associated with answers drawn from raw texts, and they also reduce the necessity to formulate new, refined, and more complex questions. For example, when one is interested in French causatives, it is frustrating to list all the inflected forms of the verb *faire* in a simple query, and many of the answers are not relevant because they involve the homonymous noun *fait* (which is also part of a lot of compounds *en fait, de fait, du fait que*, etc.). Lemmatized tagged texts are thus helpful

but inquiries about subject inversion or agentless passives are impossible to perform on corpora tagged only with parts-of-speech information. This is why people started adding phrase structure information or dependency relation to the corpora, building new types of corpora commonly called “treebanks.”

Treebanks are structurally annotated corpora that represent (in addition to part of speech and other morphological annotation) syntactic, semantic, and sometimes even intersentential relations. The word *tree* refers to the typical or base structure of the annotation, which corresponds to the notion of “tree” as defined in the formal graph theory. The interpretations of edges in the tree differ substantially among various treebanks, but they almost always represent syntactic (or more generally, structurally grammatical) relations as defined in the annotation design.

Treebanks enable linguists to ask new questions, about the word order or the complexity of various types of phrases. Arnold et al. (2000), for example, use the Penn Treebank (Marcus et al. 1993) to determine which factors favor the noncanonical V PP NP order in English. One can also check psycholinguistic preferences, in the sense that a highly frequent construction can be claimed to be preferred over a less frequent one. For example, Pynte and Colonna (2000) have shown on experimental reading tests that if a sequence of two nouns is followed by a relative clause in French, the relative clause tends to attach to the first noun if it is long and to the second noun if it is short. This claim can be easily checked on a treebank, where such a correlation can be measured (cf. Abeillé et al. 2003).

The first treebanks were, for English, the IBM/Lancaster Treebank (Leech and Roger 1991), the Penn Treebank (Marcus et al. 1993), and the Susanne corpus (Sampson 1995). At present, treebanks with different degrees of complexity are available for several languages, such as Bulgarian (Simov et al. 2002), Chinese (Sinica Treebank: Chen et al. 2003, Penn Chinese Treebank: Xia et al. 2000), Czech (PDT, Böhmová et al. 2003), Dutch (van der Beek et al. 2001), several additional treebanks for English (ICE-GB: Nelson et al. 2001, Redwood Treebank: Oepen et al. 2002a,b, etc.), French (Abeillé et al. 2003), German (Negra: Brants et al. 2003), Italian (Bosco et al. 2000, Delmonte et al. 2007), Spanish (Civit and Martí 2002, Moreno et al. 2003), Swedish (Jäborg 1986), Turkish (Oflazer et al. 2003), to name just a few.*

8.4.2 An Example: The Penn Treebank

The Penn Treebank (Marcus et al. 1993) is currently the most cited and used treebank in the world. It originally consisted of over 4.5 million words of American English. Part of the corpus was annotated for part-of-speech information and, in addition, for skeletal syntactic structure. The POS tags (comprising 36 POS tags and 12 other tags for punctuation and currency symbols) were assigned first automatically and then revised by human annotators, and the same strategy (automatic preprocessing and manual correction) holds true for the syntactic annotation. The syntactic tagset (comprising 14 tags) was similar to that used by the Lancaster Treebank Project (Leech and Roger 1991), with one important difference: the Penn Treebank scenario allowed for addition of null elements in case of some specific cases of surface deletions (such as “understood” subject of infinitive or imperative, zero variant of *that* in subordinate clauses, trace-marked positions where an *wh*-element is interpreted, and marking positions where preposition is interpreted in the so-called pied-piping contexts). The reconstruction of some of these types of “null” elements was extremely important from the viewpoint of the future plans of adding predicate-argument structures (e.g., to be able to determine verb transitivity). In a later version of the Penn Treebank, functional tags have been added to the syntactic labels (such as -SBJ, -OBJ; Marcus et al. 1994). Data in the Penn Treebank are stored in separate files for different layers of annotation (morphological, syntactical) in Lisp-like format. A graphical representation of a sample tree from the Penn Treebank, representing the sentence *A few fast-food outlets are giving it a try.*, is in Figure 8.2.

* For a more comprehensive and detailed list, see <http://faculty.washington.edu/fxia/treebank.htm>.

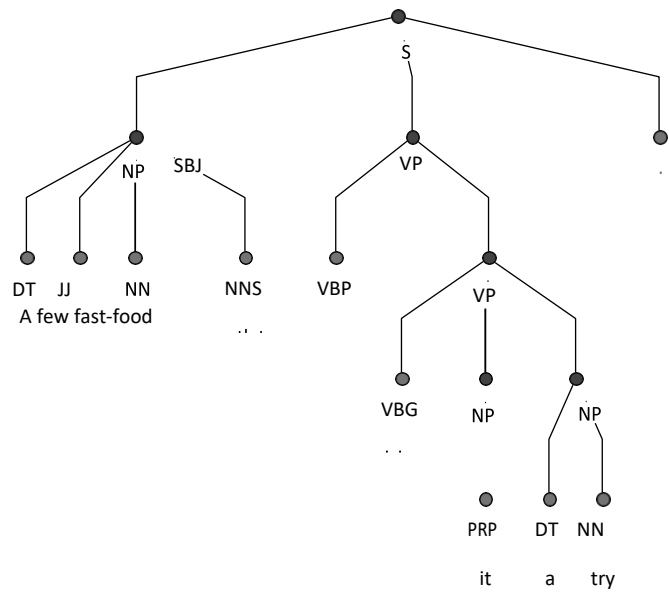


FIGURE 8.2 Example of a Penn Treebank sentence.

8.4.3 Annotation and Linguistic Theory

As already stated in the Introduction to this chapter, one of the prerequisites for achieving a reliably annotated corpus is to base the annotation scenario on a well-defined linguistic theory. This attitude has resulted in several annotation schemes based on different theoretical approaches; if these approaches are consistent, then also the annotation can be tested for its consistency. The confrontation of linguistic hypotheses with actual data leads also to checking and enriching the chosen descriptive framework.

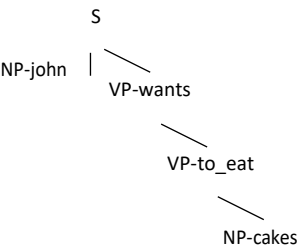


FIGURE 8.3 A simplified constituency-based tree structure for the sentence *John wants to eat cakes*.

An ongoing debate, inspired by discussions in theoretical linguistics, concerns the choice between constituency-based annotation and dependency-based annotation. As an example, let us take the English



sentence *John wants to eat cakes*. In terms of constituency, a simplified structure is given in Figure 8.3: the sentence is divided into two constituents, the noun phrase NP and the verb phrase VP, the verb phrase is in turn divided into two smaller constituents, etc. In Figure 8.4, there is a simplified dependency structure for the same sentence, with the verb as the governor of the whole structure and the SUBj(ect) and OBJ(ject) depending on it; the word *cakes* depends as the OBJ(ect) on the verb *to eat*.

Both types of annotations have their advantages and their drawbacks: con-

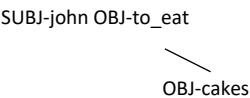


FIGURE 8.4 A simplified dependency-based tree structure for the sentence *John wants to eat cakes.*

stituents are easy to read and correspond to common grammatical knowledge (for major phrases) but they may introduce an arbitrary complexity (with more attachment nodes than needed and numerous unary phrases). Dependency links are more flexible and also correspond to common knowledge (the grammatical functions) in the sense that syntactically and semantically related nodes (words) are linked directly but consistent criteria have to be determined for the choice of the head in any group.

It is interesting to note that one of the hypotheses presented at the workshop “How to Treebank?” held at the NAACL international conference in 2007* was that given a rich enough phrase structure as well as dependency structure annotation, they can be converted to the other type of representations automatically. Which representation to choose is then supposed to be an empirical issue.

From the range of theories based on phrase-structure formalisms, let us mention, for example, the HPSG-based treebank for Bulgarian (BulTreeBank, Simov 2001, Simov et al. 2001, Slavcheva 2002), for Polish† (Marciniak et al. 2003), the LinGO Redwoods HPSG Treebank for English‡ (Oepen et al. 2002a,b, Toutanova and Manning 2002, Toutanova et al., 2002, Velldal et al. 2004), and the HPSG-oriented treebanks TuBa-E/S for English§ and TuBa-J/S for Japanese¶ (Kawata and Barteles 2000). The TuBa-E/S annotation scheme distinguishes three levels of syntactic constituency: the lexical level, the phrasal level, and the clausal level. In addition to constituent structure, annotated trees contain edge labels between nodes. These edge labels encode grammatical functions (as relations between phrases) and the distinction between heads and non-heads (as phrase-internal relations). The TuBa-J/S is a manually annotated corpus of spontaneous dialogues containing approx. 18,000 sentences (160,000 words) and the annotation scheme (similar to that of the TuBa-E/S corpus) is enriched by the establishment of a label assigned to the root node and determining the type of the sentence. This scheme also includes additional tags to capture the specific phenomena for speech.

As an example of a dependency-based annotation scheme, let us mention the PDT. The PDT (see e.g., Hajič 1998, Böhmová et al. 2003) consists of continuous Czech texts (taken from the Czech National Corpus) analyzed on three levels of annotation (morphological, surface syntactic shape, and underlying syntactic structure). At present, the total number of documents annotated on all the three levels is 3,168, amounting to 49,442 sentences and 833,357 (occurrences of) nodes. The PDT Version 1.0 (with the annotation of the first two levels) is available on CD-ROM as well as the present Version 2.0 (with the annotation of the third, underlying level). One of the important distinctive features of the PDT annotation is the fact that in addition to the morphemic layer and to the annotation of the surface shape of the sentences the scenario includes annotation on the underlying (tectogrammatical) layer (see, e.g., Figure 8.5). The underlying sentence structure is captured in the form of a dependency tree representing (one of) the (literal) meaning(s) of a sentence. Only autosemantic words are represented as nodes of the tree, function words having indices of node labels as their counterparts on this level (among these, the functors represent the dependency relations, i.e., arguments and adjuncts, and the values of grammatemes represent morphological units such as tenses, numbers, modalities, etc.). New nodes (not present in the morphemic form of the sentence) are added to account for surface deletions. Each of the edges of the tree instantiates one type of dependency (more exactly, dependency can be understood as a set of binary relations, i.e., of arguments and adjuncts; certain technical adjustments have been necessary for including the relations of coordination, apposition, and parenthesis). In the valency frame of the head word (contained in its lexical entry), it is specified which arguments and adjuncts are obligatory with this word. Each node of the dependency tree structure is labeled not only by its underlying function (e.g., the function of an argument as Actor, Addressee, Patient, Effect, Origin, or adjuncts such as one of the types of Locatives and of Temporal modification, or Cause, Accompaniment, Manner, etc.), but also by one of the three values (c, t, f) of the attribute of information structure, namely contrastive contextually bound node, contextually bound node, and contextually non-bound node, in that order (see Hajičová and Sgall 2001, Hajičová 2002, Veselá et al. 2004). Figure 8.5 presents an example of the annotation of the Czech sentence *Česká opozice se nijak netají tím, že pokud se dostane k moci, nebude se deficitnímu rozpočtu nijak bránit.* (lit.: Czech opposition Refl. in-no-way keeps-back-the-fact that in-so-far-as [it] will-come into power, [it] will-not Refl. deficit budget in-no-way oppose. English translation: *The Czech*

* <http://faculty.washington.edu/fxia/treebank/workshop07/agenda.htm>

† <http://nlp.ipipan.waw.pl/CRIT2>

- # <http://wiki.delph-in.net/moin/RedwoodsTop>
- § http://www.sfs.uni-tuebingen.de/en_tuebaes.shtml
- ¶ http://www.sfs.uni-tuebingen.de/en_tuebajs.shtml

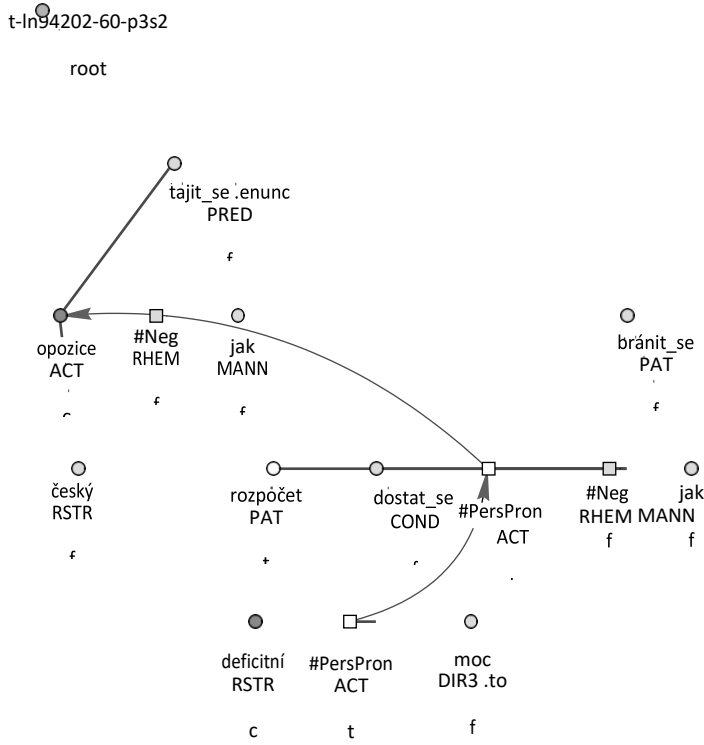


FIGURE 8.5 A sample tree from the PDT for the sentence: *Česká opozice se nijak netají tím, že pokud se dostane kmoci, nebude se deficitnímu rozpočtu nijak bránit.* (lit.: Czech opposition Refl. in-no-way keeps-back the-fact that in-so-far-as [it] will-come into power, [it] will-not Refl. deficit budget in-no-way oppose. English translation: *The Czech opposition does not keep back that if they come into power, they will not oppose the deficit budget.*)

opposition does not keep back that if they come into power, they will not oppose the deficit budget.) There are three complementations of the main verb *tajit se* (keep back), namely Actor, Manner and Patient, and a negative rhematiser, and the governing verb of the dependent clause *bránit se* (oppose) depends on the main verb as its Patient and has five complementations and negation depending on it; the arrow leading from the reconstructed Actors to the node *opozice* (opposition) reflects the coreference relation. These coreference relations may, of course, cross the sentence boundary.

Another example of a treebank that includes (some kind of) dependency information is the French Treebank* with morphosyntactic and constituent annotations (more or less compatible with various syntactic frameworks) and also annotations of grammatical functions associated with major constituents which depend on a Verb or a Verbal Noun (Abeillé et al. 2003). See Figure 8.6 for a (simplified) graphical display of a sample sentence from the French Treebank.

Also the treebanks of Scandinavian languages belong to the family of dependency treebanks, see the Nordic Treebank Network,[†] which is related to treebanks in the Nordic countries developed in cooperation with several Scandinavian universities and including the national corpora of written and/or spoken language annotated manually or semiautomatically (see e.g., Nivre 2002 for Swedish, Bick 2003 and Kromann 2003 for Danish). Among the dependency-based treebanks the Turin University Italian

Treebank should be mentioned (Bosco 2000) as well as the treebanks of Turkish (Oflazer et al. 2003), of Basque (Aduriz et al. 2003) and of Greek (Prokopidis et al. 2005).

* <http://www.llf.cnrs.fr/Gens/Abeillé/French-Treebank-fr.php>

† <http://w3.msi.vxu.se/~nivre/research/nt.html>

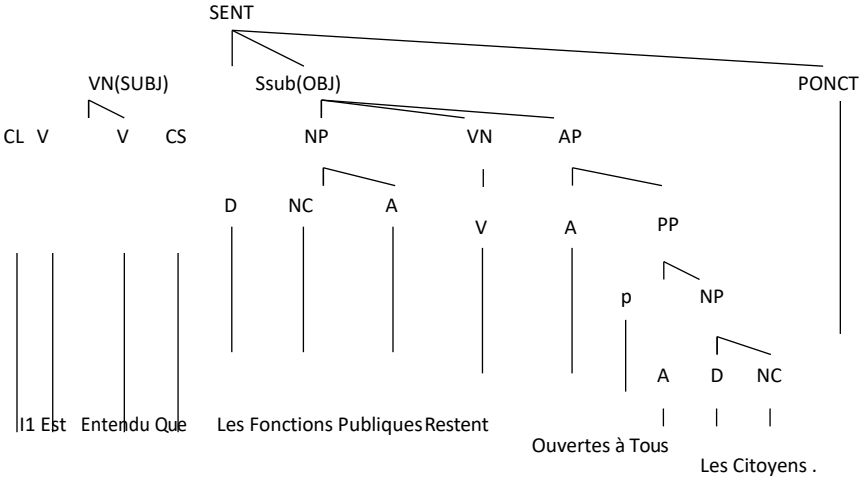


FIGURE 8.6 A sample French tree. (English translation: *It is understood that the public functions remain open to all the citizens.*)

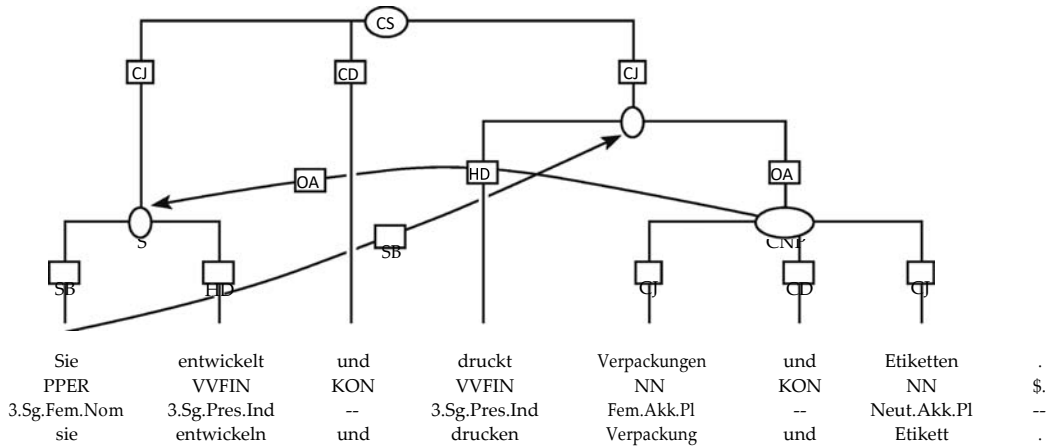


FIGURE 8.7 Example from the Tiger corpus: complex syntactic and semantic dependency annotation. (English translation: *It develops and prints packaging materials and labels.*)

A hybrid solution, chosen by some of the projects, is to maintain some constituents with dependency relations between them (cf. Brants et al. 2003, Chen et al. 2003, Kurohashi and Nagao 2003, Oflazer et al. 2003). In this case, the constituents are usually minimal phrases, called chunks, bunsetsu (Japanese), or inflection groups (Turkish). In addition to the constituent structure, annotated trees contain edge labels between nodes encoding grammatical functions and the distinction between heads and non-heads. For a typical case, see Figure 8.7, which is an example from the German Tiger Treebank (Brants et al. 2003).

The choice of annotation depends both on the availability of syntactic studies for a given language (formalized or not, theory oriented or not) and on the objective. Carroll et al. (2003) demonstrate how difficult it is to choose a reasonable annotation scheme for parser evaluation, when a variety of parsing outputs have to be matched with the annotated corpus.

8.4.4 Going Beyond the Surface Shape of the Sentence

The development of annotation schemes for large text corpora entered a new stage passing over from scenarios with tags covering phenomena “visible” or “transparent” on the surface shape of the sentence (be they of morphological or shallow syntactic character) to some kind of an underlying structure of

the sentence; moreover, not only intrasentential but also intersentential relations are paid attention to. Naturally, underlying layer schemes also have to tackle ellipsis resolution, or in other words, the reconstruction of the items deleted on the surface shape of the sentences.

The urgency of deep tagging was emphasized, for example, by Uszkoreit (2004) and some interesting work in this direction is documented by the recent development of several treebanks, for example, Penn PropBank based on the Penn Treebank for English; see Kingsbury and Palmer (2002) with a related project of NomBank* the task of which is to mark the sets of arguments that co-occur with nouns in the PropBank Corpus, just as PropBank records such information for verbs (Meyers et al. 2004). The first large treebank based on a consistent account of underlying dependency relations was the tectogrammatical layer of the PDT for Czech (with the current first steps in testing the scheme for English, German, or even Arabic), see Hajičová (2002), followed by the Tiger/Negra project for German[†] (Brants et al. 2002) or the Redwood Treebank (Oepen et al. 2002). It should be noted that a common feature of all these proposals is an annotation scheme coming close to predicate argument structure. This is also a distinctive feature of the FrameNet project of semantic roles labeling (see e.g., Fillmore et al. 2002, 2003, Palmer et al. 2005) that has been originally developed for English but is being extended to other languages such as German within the SALSA project (Burchardt and Frank 2006), Bulgarian (Balabanova and Ivanova 2002), Dutch (Monachesi et al. 2007), etc. However, it should be noted that FrameNet does not link to any real-world text(s) or corpora—it only extracts relevant examples to the frames it consists of.

It is now commonly accepted in theoretical linguistics that one of the semantically relevant aspects of sentences is their information structure (topic-focus articulation). It is a great challenge for corpus annotation projects to develop a scenario that would reflect also this structuring; for a possibility, see the topic-focus annotation of the PDT (e.g., Hajičová 2003); for a rather recent attempt to annotate a spoken corpus as for the information structure, see Calhoun et al. (2005).

A related issue, the tackling of which has to go beyond the boundaries of the sentence, or, in more general terms, to take into account a broader context, is that of anaphora resolution, which means an establishment of anaphoric (coreference) links between individual referential expressions. This is a widely discussed issue in theoretical writings but the work on establishing such links in larger corpora is still rare (e.g., see adding pronoun-antecedent links in Fligelstone (1992), Tutin et al. (2000) or in the scenario of the PDT).

Considerable attention is now being paid to the analysis of intersentential relations leading to a build-up of discourse treebanks. The biggest and in a sense a pioneering comprehensive project is that of the Penn Discourse Treebank[‡] (PDTB); the first initiative to turn attention to discourse relations and discourse markers can be found in Webber and Joshi (1998) in relation to Joshi's theory of the Lexicalized Tree-Adjoining Grammar. The project has resulted in a large-scale corpus annotated with information related to discourse structure, focusing on encoding coherence relations associated with discourse connectives. The annotations include the argument structure of the connectives, basic sense distinctions for discourse connectives, and the assignment of attribution-related features for both connectives and their arguments (more recently, see e.g., Prasad et al. 2007, Miltsakaki et al. 2008, Prasad et al. 2008, Pitler et al. 2008). Senses are set up hierarchically, distinguishing four classes (Temporal, Contingency, Comparison, and Expansion); these classes are divided into several types and subtypes. The hierarchical structure helps the inter-annotator agreement. Experiments are carried out in the areas of statistics on the corpus, automatic text summarization, predicting discourse coherence, and also in automatic discourse annotation.

A similarly oriented project is that of an extension of the annotations of the PDT beyond the boundaries of sentences (see Mladová et al. 2008). The scenario is conceived of as based on the tectogrammatical annotations, making use of some of the analyses present already there (a special functor is included in the scenario for general, semantically non-distinguished intersentential relationships and also the

* <http://nlp.cs.nyu.edu/meyers/NomBank.html>

[†] <http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERCorpus>

[‡] <http://www.seas.upenn.edu/~pdtb>

intrasentential relationships of different kinds of coordination and embedded clauses are being made use of).

Another stream of semantic annotation, going often hand in hand with the deep syntactic annotations, concerns word sense disambiguation. The development of the original WordNet (Fellbaum 1998) and its language-specific derivatives gave rise to great expectations for a progress in sense disambiguation and also had a strong impact in the field of semantic lexicons. An attempt is being made to integrate the two semantic lexical resources in one of the projects of the Pisa research center, namely the ItalWordNet and the Italian SIMPLE lexicon (Roventini et al. 2002).

There is a great variety of annotation schemes focusing on some specific linguistic or extralinguistic phenomena: some of them are apparently easier to handle (such as temporal, spatial, or manner relations), some are more difficult to capture and the scenarios are still in an experimental stage (such as treatment of opinions: good/bad judgments, true/false beliefs, speech acts, presuppositions, and metaphors).

The range of these proposals only documents that there is a constant hunger for annotated corpora. However, corpus annotation is a demanding task in many aspects: it is a time consuming and a very expensive activity, and therefore ways are being investigated how to achieve a large amount of annotated data with least effort. One such possibility is being explored by Hladká (Hajičová and Hladká 2008); in her project, she aims at a design, the implementation, and the evaluation of a game-playing system as an alternative way of generating linguistic data needed by the tasks of coreference resolution, named entity recognition and document labeling. The system is designed to use the “Games with a Purpose” methodology originally proposed for image labeling and is supposed to be language independent.

8.5 The Process of Building Treebanks

In some of the first projects, corpus annotation was done from scratch entirely by humans, for example, in Sweden (Jäborg 1986), or for the Susanne corpus (Sampson 1995). Purely manual annotation is done by Marciniak et al. (2003). More often, the annotation is at least partially automated; however, automatic annotation—even if of high quality—is not always desirable since it is the human (linguistic) interpretation of a new material that is crucial to the future use of the treebank, especially for training statistical parsers and other analyzers from treebanks.

Automatic tools for corpora annotation (such as taggers, parsers, etc.—see Chapter 10 and the subsequent ones) exist for many languages, but obviously, they make mistakes. One could, in principle, run some automatic part-of-speech tagger or lemmatizer on a given corpus, and use the resulting annotation. While it might be adequate for searching, the quality of the resulting corpus is not guaranteed. In any case, the state-of-the-art automatic tools learn on manually annotated corpora so that manual annotation plays and will always play an important role in the whole process. In such corpora, annotations are devised by experienced linguists or grammarians, fully documented for the end user, and fully checked to minimize the remaining errors (introduced by a human annotator or reviewer).

Human post-checking is always necessary, be it systematic (as in the Brown Corpus for English, in the Penn Treebank or in the PDT) or partial (as in the British National Corpus). Some check all annotation levels at the same time (Brants et al. 2003, Chen et al. 2003), others check tagging and parsing information separately (Penn Treebank, Abeillé et al. 2003, Böhmová et al. 2003). Semantic information is often annotated by hand. Large-scale annotation projects typically involve dozens of human annotators, and ensuring coherence among them is a crucial matter as pointed out by Wallis (2003) for the ICE-GB project, or Brants et al. (2003) for the Negra project.

Several methods have been proposed to check the inter-annotator agreement, even for the semantic annotation and the discourse annotation that are less standardized than the morphosyntactic ones. They are based on sophisticated statistical techniques; a simple percentage agreement is insufficient because it does not discriminate how much agreement is obtained just by chance. Coefficients *S* (Bennett et al.

1954), π (Scott 1955), and κ (Cohen 1960) all measure the agreement between two annotators obtained

above chance. The κ (kappa) coefficient is the most general and the most widely used today

$$\kappa = \frac{p(A) - p(C)}{1 - p(C)},$$

where

$p(A)$ is the percentage (or ratio) where the annotators have agreed

$p(C)$ is the probability that they agreed just by chance

In fact, the other agreement coefficients (S , π) only differ in the way they estimate the level of a chance agreement between the two annotators. Generalizations of these coefficients for more than two annotators also exist. To take the chance agreement into account, it is necessary to have prior knowledge of the distribution of the different possible annotations, which is difficult in natural language, especially for tasks such as anaphora annotation, for which there is no prior list of tags. Carletta (1996) has proposed to use a K coefficient of agreement, which takes into account annotators' biases. Krippendorff's α (Krippendorff 1980, 2004) makes it possible to differentiate degrees of disagreement, since some disagreements may be more serious than others. It is difficult to interpret values of agreement coefficients and there is a lack of consensus on it. Overall, an agreement at least 0.8 is considered to ensure high annotation quality. The number of possible tags and the number of annotators can play a crucial role in computing the agreement and sometimes a coefficient above 0.7 is good enough. Arstein and Poesio (2008) give an excellent survey of methods for measuring agreement among corpus annotators; see also (Eugenio and Glass 2004).

An attempt to propose a method for error detection and correction in corpus annotation is presented by Dickinson, first in his PhD dissertation in 2005* and then within the DECCA (Detection of Errors and Correction in Corpus Annotation[†]) project. Dickinson's method relies on the recurrence of identical strings with varying annotation to find erroneous mark-up. The method can be most readily applied to positional annotation, such as part-of-speech annotation, but can be extended to structural annotation, both for tree structures (as with syntactic annotation) and for graph structures (with syntactic annotation allowing discontinuous constituents, or crossing branches). The results indicate that errors are detected with 85% accuracy (see Dickinson 2006a,b, Boyd et al. 2007).

Most treebanks involving human validation are long and costly enterprises. In order to minimize time and cost, some new projects tend to favor merging different outputs of automatic annotation tools such as tagging with different taggers and checking only the parts with conflicts, for example, in the French Multitag project, cf. Adda et al. (1999).

Many data formats have been employed for encoding treebanks. The choice depends on the complexity of the treebank and its expected usage. Some formats can be easily read by humans—CSV (CoNLL Shared Task), simple usages of SGML (PDT 1.0), others are more easily read by computer programs—Lisp-like bracketing style (Penn Treebank, PDT 1.0), XML (PDT 2.0, French Treebank), database storage (Tiger Treebank). XML-based standards for encoding text corpora have been proposed and used, for example, XCES (Ide et al. 2000), Annotation Graphs (Cieri and Bird 2001), or PML (Pajas and Štěpánek 2008).

8.6 Applications of Treebanks

Corpus annotation is not a self-contained task: it serves, among other things, as

1. A training and testing data resource for NLP.
2. An invaluable test for linguistic theories on which the annotation schemes are based.
3. A resource of linguistic information for the build-up of grammars and lexicons.

* <http://ling.osu.edu/dickinso/papers/diss/>

† <http://decca.osu.edu/index.html>

The main use of treebanks is in the area of NLP, most frequently and importantly statistical parsing.* Treebanks are used as the training data for supervised machine learning (which might include the automatic extraction of grammars) and as test data for evaluating them. Recently, as corpora that combine syntactic and semantic annotation have become more widely available, parsers are also trained jointly with semantic role labelers (for different types of parsers and their evaluation, see e.g., Charniak 1993, Charniak 1996, Collins 1997, Bod 1998, Carroll 1998, Xia et al. 2000, Bod 2003, Carroll et al. 2003, Chen and Vijay-Shanker 2004, McDonald and Pereira 2006, Nivre 2006, Surdeanu et al. 2008; for a broad and comprehensive description of parsing, see Chapter 11). In essence, to find the correct analysis (the best parse) amounts to finding such a derivation tree that maximizes the probability (or “score” in general) of that derivation (usually the product of the probabilities or scores of the rules being used). For dependency trees, the situation is analogous, except that edges and their probabilities (scores) are used instead of rules. The probabilities (or scores) are estimated from the treebank during the training phase of the machine learning method used (and smoothed accordingly). Such parsers are robust — they do not fail on real corpora, as grammar-based parsers often did in the past when written manually without a proper use of scoring or probabilities. They can give just one best result, or they can provide more results, sorted according to their probability (or score).

Treebanks serve one more important purpose with regard to parsing: they are used for their evaluation. When performing such evaluation, one has to divide the corpus into a training and development part, which is used for the extraction, estimation, smoothing, and any other “tuning” of the parser’s probabilities or scores; and an evaluation part, which is used only for determining the parser’s accuracy after its development is completely finished. Such a division is necessary to simulate a real-world situation when the parser is applied to previously unseen data. The training/evaluation data division is typically performed several times so that the evaluation part is different for every training/evaluation experiment (called the “cross-validation” of results), to avoid biases caused by using only a single section of the treebank for evaluation.

Nevertheless, treebanks tend to be consistent in their domain, and parser performance on out-of-domain texts is usually worse than in an experiment that uses carefully simulated real-world but still in-domain evaluation data. Parser adaptation and its relation to treebank annotation is thus subject to current research. For example, the Penn Treebank’s *Wall Street Journal* part, which is traditionally used as training and evaluation material for statistical parsers, comes almost exclusively from the financial domain, despite an occasional article on a new book, film or sports event; it is thus no wonder that a cross-treebank evaluation against another English treebank yields significantly worse results.†

The type of parser follows the treebank style (dependency parsers are trained on dependency treebanks and phrase-based treebanks are used for parsers based on context-free grammars), unless the treebank is converted first to a treebank of the other type (for a discussion how and when is such conversion possible, see Section 8.4.3).

It is also possible to merge treebanks with other annotated data, such as predicate-argument relations and valency, named entities, coreference annotation, etc. (provided they have been done on the same data). Such combined resources are very useful for applications where parsing and other types of classification are to be performed simultaneously or even jointly.‡

To train a system that performs well, one has to find a balance between the size of the tagset (richness of information available in the annotated corpus) and the size of the corpus (number of words or sentences annotated). Relatively good performances are obtained with a small tagset (less than 50 tags) and a large corpus (more than 1 million words). Srinivas and Joshi (1999), training a shallow parser

* Treebanks can also be used (and often are) for training statistical taggers, since they commonly contain part-of-speech and morphological annotation as well. For the description of tagging approaches, see Chapter 10.

† See e.g., the CoNLL-2007 and CoNLL-2009 Shared Task out-of-domain adaptation results (<http://nextens.uvt.nl/>)

[depparse-wiki/AllScores](#) and <http://ufal.mff.cuni.cz/conll2009-st/results/results.php>).

[‡] See again the CoNLL-2008 and CoNLL-2009 Shared Task descriptions at <http://barcelona.research.yahoo.net/conll2008> and <http://ufal.mff.cuni.cz/conll2009-st>

(called a supertagger) on the Penn Treebank, show how going from a training set of 8000 words to a training set of 1 million words improve the parser's performance from 86% (of the words with the correct syntactic structure) to 92%. The size of the various tagsets (label inventories) is determined at the treebank annotation time, but downsizing is always possible once the treebank is used as data for a particular training method.

Automatically trained statistical parsers are not the only (even if prevailing) use of treebanks.

Attempts to combine manual grammar development with semiautomatic corpus inspection and grammar rule extraction are still on. Rosén et al. (2006) notice that the manual syntactic annotation of corpora is a good empirical source for grammar development, as opposed to introspection and constructed examples, but an automatic syntactic annotation of corpora, which is fast and always consistent, requires a fully adequate grammar, which in turn should ideally be based on a corpus. The authors propose to break this seemingly vicious circle by an incremental approach that closely links grammar development and treebank construction.

As observed by Charniak (1996), treebank grammars, as a simple list of context-free rewrite rules, are often much larger than human-crafted ones (more than 10,000 rules for the Penn Treebank) and may decrease the efficiency of the parser. This is why some authors (Chen and Vijay-Shanker 2004, Xia et al. 2000) start with a linguistic model of grammar (LFG, HPSG, or TAG) that guides the type of pattern (tree-like or rule-like) to be extracted.

Other applications include text classification, word sense disambiguation, multilingual text alignment, and indexation. For text categorization, Biber (1988) works with richly annotated texts, and uses criteria such as the relative proportion of adjectives among categories (as a good discriminator for formal vs. informal speech), and the relative proportion of pronominal subjects among all subjects (as a discriminator for speech vs. written language). Malrieu and Rastier (2001) duplicate such criteria on other languages such as French.

The first corpus annotated with word senses was SemCor (Landes et al. 1998), a subset of the Brown Corpus annotated with senses from WordNet. It was developed together with (and for the purpose of) an improved WordNet (Version 1.6). New word senses identified in the data were added into the WordNet during the annotation.

In the PDT, two major word-sense-related annotation experiments took place: in 2006, a small portion of the PDT 2.0 was annotated with senses (synsets) from the Czech WordNet. The annotation proceeded word-after-word, but unlike in the case of SemCor, words that had no meaning in the Czech WordNet were skipped, that is, the annotation lexicon was not modified during the annotations. Each word was annotated individually. The other annotation took place later, and added the identification and sense-annotation of multi-word expressions in the PDT 2.0. Multi-word named entities were annotated as well. This later project was similar to the SemCor one in the handling of the annotation lexicon: the project started with a lexicon containing multi-word expressions compiled from several dictionaries, but new expressions (and meanings of existing expressions) were added as they were identified in the data.

For verb sense disambiguation, it is important to know the context of each occurrence, since it is a good guide for semantic interpretation (*can* as a modal *can* be distinguished from *can* as a transitive verb, etc.). For automatic word sense classification, parsed texts are also being used, for example, by Merlo and Stevenson (2001).

For text indexing or terminology extraction, too, some syntactic structure is necessary, at least for spotting the major noun phrases. Knowing that an NP is in argument position (subject or object) may make it a better candidate for indexing than NPs only occurring in adjunct phrases.

(Manually) treebanked texts are usually much smaller than raw texts. While the latter are usually available in quasi-infinite quantity (especially via various Web sites in the world as well as from established language data publishers, such as the Linguistic Data Consortium* and ELDA[†]), the former often require

* <http://www ldc.upenn.edu>

† <http://www.elda.org>

human post-correction, without which they do not perform well enough for many languages. Some searches for individual forms or patterns may yield poor results. Another obstacle is that treebanks are not readable as such and require specific search and viewing tools. This may be why they are still more used by computational linguists than by other linguists.

It is quite understandable that together with the rapid increase of the number of languages for which annotated corpora are being developed, an agreement on some standards is felt as a priority, important also for the possibility of national or international cooperation. Among the existing standards, there are such that have been consensually agreed in multilingual initiatives such as EAGLES/ISLE, or de facto standards such as (Euro)WordNet or PAROLE/SIMPLE. The trend toward standardization brings about several important issues to be discussed. One of them is the possibility/impossibility of establishing some “theory neutral” annotation scheme. “Theory neutral,” of course, does not mean “without any theoretical background”; it is our firm conviction that a reliable and meaningful scheme of annotation must be backed by a solid, empirically verified, and theoretically sound linguistic framework. “Theory neutral” may only be understood as an attempt to develop an annotation scenario on some underlying level of annotation that would be applicable to any language, or at least to languages that are typologically close to each other (see also Nivre 2003).

Another issue connected with the standardization efforts is that of the translatability of one annotation scheme to another; this feature is recently referred to as an interoperability of annotation schemes. The task is very important especially for an evaluation of annotated corpora for most different languages and its feasibility can be, for example, documented by the efforts to translate the Penn PropBank (developed from the Penn Treebank 2) argument structures into the dependency based underlying (tectogrammatical) structures of the PDT and vice versa (Žabokrtský and Kučerová 2002) and also the comparative studies carried out for these two approaches and for the UMC Lexical Database based primarily on Levin’s (1993) classification of verb frames. Another attempt of a similar kind is the description of an algorithm translating the Penn Treebank into a corpus of Combinatory Categorical Grammar normal-form derivations (Hockenmeier and Steedman 2002), the tool described in Daum et al. (2004) for converting phrase treebanks to dependency trees or the discussion of the portability of methods and results over treebanks in different languages and annotation formats in Bosco and Lombardo (2006).

8.7 Searching Treebanks

The ability to search treebanks is crucial for linguists and important for computer scientists, albeit the latter use it mainly for inspecting the corpus and finding features important for building appropriate statistical models that they learn from them automatically.

Linear corpora (plain or tagged) can be very large (billions of tokens) and search tools for them have to be optimized to work with such extremely large data. Manatee/Bonito* is a client-server application that uses regular expressions and Boolean expressions to create complex queries on linearly annotated texts. The search results can be processed further to obtain frequency lists and other statistical data. The Stuttgart Corpus Workbench[†] is a collection of tools for searching plain or tagged texts. It uses a similar query language such as Manatee/Bonito and also can further process the search results.

Most corpora referred to here are static resources. A recent line of research is to develop dynamic treebanks, which are parsed corpora distributed with all annotation tools, in order to be enlarged at will by the user (cf. Oepen et al. 2002).

A treebank query language is only as good as the list of linguistic phenomena it offers to be used easily (or at all) in the searches. It has been noted (Kallmeyer 2000, Cassidy 2002, Lai and Bird 2004, Bird et al. 2006, Mírovský 2008a) that the standard XML query languages (such as XQuery that in turn builds on

* <http://www.textforge.cz/products>

[†] <http://www.stanford.edu/dept/linguistics/corpora/cas-tut-cqp.html>

XPath, Clark and DeRose 1999) cannot deal with some of the linguistic phenomena at all, or only in a way that is unacceptable for the users. In Bird et al. (2005), three expressive features important for linguistics queries are listed: immediate precedence, subtree scoping, and edge alignment.

Many search tools for treebanks have been developed so far. TGrep (Pito 1994) is a traditional line-based search tool developed primarily for the Penn Treebank (Marcus et al. 1993). It can be used for any treebank where each node is labeled with only one symbol — either a nonterminal or a leaf with an atomic token. Regular expressions can be used for matching node symbols. TGrep2 (Rohde 2005) is a sequel to TGrep. It is almost completely backward compatible with TGrep but brings a number of new features, such as full logical expressions on relationships and patterns, co-indexing (and handles cyclical links) and user-defined “macros.” TigerSearch (Lezius 2002) is a graphically oriented search tool for the Tiger Treebank (Brants et al. 2002). The query language allows for Boolean feature-value pair expressions, immediate precedence, and immediate dominance of nodes, and on the highest level, Boolean expressions over node relations (without negation). TrEd (Pajas 2007) has primarily been used for the manual annotation of the PDT and other similar treebanks with some perl-based general search language. TrEd now contains a structured and fast, client-server, end-user-oriented extension “Tree_Query,” which efficiently implements (both in online- and batch modes) most of the known linguistic query requirements on any PML-encoded corpus (Pajas and Štěpánek 2008). Netgraph (Mírovský 2008b) is a powerful, graphically oriented client-server based search tool, primarily developed also for the PDT 2.0. It uses meta-attributes, node co-indexing, and arithmetic and logical relations to express some of the necessary features that are not easily expressible by immediate relations in queries. Vigtorya (Steiner and Kallmeyer 2002) is a search tool developed for the Tübingen Treebanks (Hinrichs et al. 2000). It has a graphical interface, but without a visual depiction of the query. A first order logic without quantification is chosen as a query language, with some restrictions. Another query language developed for the Tübingen Treebanks is the Finite structure query (fsq, Kepser 2003). It uses the full first-order logic (with quantification), with LISP-like syntax.

8.8 Conclusions

Treebanking is a highly complex issue and many questions still remain to be discussed and resolved. At the 2007 NAACL workshop on treebanking mentioned earlier, the following three general topics followed by language-specific issues were discussed as being topical for the present-state-of-the-art:

1. Lessons learned from the Penn Treebank methodology. (What semantics is to be annotated? What information was missing in the underlying Penn Treebank? What information was there but represented badly? What methodology is appropriate for semantic annotation? What are the advantages of a phrase-structure and/or a dependency treebank for parsing as such and especially for semantic annotation?)
2. Grammar formalisms and transformations between formalisms (including the pros and cons for building a treebank for grammars in a particular formalism vs. building a general purpose treebank and extracting grammars from the Treebank).
3. Treebanks as training data for parsers, tackling such issues as whether a more refined tagset for parsing is preferable, which categories are useful and which are not, or what are the advantages and disadvantages of automatic preprocessing of the data to be treebanked.

Given the current state of syntactic knowledge, some annotation choices may be arbitrary. What is most important is consistency (similar cases must be handled similarly) and explicitness (a detailed documentation must accompany the annotated corpus). As noted by G. Sampson (1995), for the Susanne Corpus, the size of the documentation can be bigger than the corpus itself. Without consistency any annotated corpus becomes useless; at the same time, any annotation (except in case of a fully automatic annotation which, regretfully, seems to be far to be achieved with some reasonable richness of annotation

labels and categories) involves some human intervention and as such is open for inconsistencies. Therefore,

the agreement between annotators should be carefully watched and measured, in order to make the annotation guidelines more explicit and unambiguous.

Thanks to treebanks, NLP technologies such as automatic tagging, parsing, and other annotation of (mostly) written texts has made tremendous progress during the past 10–20 years. Part-of-speech tagging seems to be close to its current limits, reaching the level of human performance (as defined by the interannotator agreement). Parsing, “deep” parsing, semantic role labeling, machine translation, and other NLP technologies are still areas of vivid research and experimentation. It is expected that the findings accumulated during these experiments will influence future treebank annotation projects to serve better NLP technology needs. Similar influence might come from the theoretical side: new annotation schemes will then support, in the areas of syntax and semantics, (hopefully) more consistent, more adequate, and more explanatory linguistic theories than they do today.

Acknowledgments

The authors acknowledge the support of the Czech Ministry of Education (grants MSM-0021620838 and ME838), the Czech Grant Agency (project under the grant 405/09/0729), and the Grant Agency of Charles University in Prague (project GAUK52408/2008). We are grateful to Barbora Vidová Hladká and Zdeněk Žabokrtský for reading and commenting upon the first draft of the chapter and for providing us with useful information and recommendations we used in the relevant places of the text, as well as to Pavel Straňák for his additions in the paragraphs on word sense disambiguation and named entities. Thanks are due to the two reviewers of the chapter Steve Bird and Adam Meyers for most helpful comments.

References

- Abeillé, A., Clément, L., and F. Toussenen. 2003. Building a treebank for French. In *Treebanks: Building and Using Parsed Corpora*, ed. A. Abeillé, pp. 165–188. Dordrecht, the Netherlands: Kluwer.
- Adda, G., Mariani, J., Paroubek, P., Rajman, M., and J. Lecomte. 1999. L’action GRACE d’évaluation de l’assignation de parties du discours pour le français. *Langues* 2(2): 119–129.
- Aduriz, I., Aranzabe, M., Arriola, J. et al. 2003. Methodology and steps towards the construction of EPEC, a corpus of written Basque tagged at morphological and syntactic levels for the automatic processing. In *Proceedings of the Corpus Linguistics 2003 Conference*, Lancaster, U.K., eds. D. Archer, P. Rayson, A. Wilson, and T. McEnery, pp. 10–11. UCREL technical paper (16). UCREL, Lancaster University.
- Arnold, J. E., Wasow, T., Losongco, A., and R. Ginstrom. 2000. Heaviness vs. newness: The effects of structural complexity and discourse status on constituent ordering. *Language* 76: 28–55.
- Arstein R. and M. Poesio. 2008. Inter-coder agreement for computational Linguistics Inter-Coder agreement for computational linguistics. *Computational Linguistics* 34(4): 555–596.
- Balabanova, E. and K. Ivanova. 2002. Creating a machine-readable version of Bulgarian valence dictionary (A case study of CLaRK system application). In *Proceedings of TLT 2002*, Sozopol, Bulgaria, eds. E. Hinrichs and K. Simov, pp. 1–12.
- Bennett, E. M., Alpert, R., and A. C. Goldstein. 1954. Communications through limited questioning. *Public Opinion Quarterly* 18(3): 303–308.
- Biber, D. 1988. *Variation Across Speech and Writing*. Cambridge, U.K.: Cambridge University Press.
- Bick, E. 2003. Arboretum, a Hybrid Treebank for Danish. In *Proceedings of TLT 2003*, Växjö, Sweden, eds. J. Nivre and E. Hinrich, pp. 9–20.

Bird, S., Chen, Y., Davidson, S., Lee, H., and Y. Zheng. 2006. Designing and evaluating an XPath dialect for linguistic queries. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, Atlanta, GA, pp. 52–61.

- Bird, S., Chen, Y., Davidson, S., Lee, H., and Y. Zheng. 2005. Extending Xpath to support linguistic queries. In *Proceedings of the Workshop on Programming Language Technologies for XML (PLAN-X 2005)*, San Francisco, CA, pp.35–46.
- Bird, S. and J. Harrington. 2001. Editorial to the special issue of Speech Communication. *Speech Annotation and Corpus Tools* 33:1–4.
- Bod, R. 1998. *Beyond Grammar*. Stanford, CA: CSLI Publications.
- Bod, R. 2003. Extracting grammars from treebanks. In *Treebanks: Building and Using Parsed Corpora*, ed. A. Abeillé, pp. 333–350. Dordrecht, the Netherlands: Kluwer.
- Böhmová, A., Hajič, J., Hajičová, E., and B. Hladká. 2003. The Prague dependency treebank: A 3-level annotation scenario. In *Treebanks: Building and Using Parsed Corpora*, ed. A. Abeillé, pp. 103–128. Dordrecht, the Netherlands: Kluwer.
- Bosco, C. 2000. A richer annotation schema for an Italian treebank. In *Proceedings of ESSLLI-2000 Student Session*, ed. C. Pilière, Birmingham, U.K., pp. 22–33.
- Bosco, C. and V. Lombardo. 2006. Comparing linguistic information in treebank annotations. In *Proceedings of LREC 2006*, Genova, Italy, pp. 1770–1775.
- Bosco, C., Lombardo, V., Vassallo, D., and L. Lesmo. 2000. Building a treebank for Italian: A data-driven annotation schema. In *Proceedings of LREC 2000*, Athens, Greece, pp. 99–105.
- Boyd, A., Dickinson, M., and D. Meurers. 2007. Increasing the recall of corpus annotation error detection. In *Proceedings of TLT 2007*, Bergen, Norway, NEALT Proceedings Series, Vol. 1, pp. 19–30.
- Brants, S., Dipper, S., Hansen, S., Lezius, W., and G. Smith. 2002. The TIGER treebank. In *Proceedings of TLT 2002*, Sozopol, Bulgaria, eds. Hinrichs, E. and Simov, K., pp. 24–41.
- Brants, T., Skut, W., and H. Uszkoreit. 2003. Syntactic annotation of a German newspaper corpus. In *Treebanks: Building and Using Parsed Corpora*, ed. A. Abeillé, pp. 73–88. Dordrecht, the Netherlands: Kluwer.
- Burchardt, A. and A. Frank. 2006. Approximating textual entailment with LFG and FrameNet frames. In *Proceedings of the Second PASCAL Recognising Textual Entailment Challenge Workshop*, Venice, Italy, ed. B. Magnini and I. Dagan, pp. 92–97.
- Calhoun, S., Nissim, M., Steedman, M., and J. Brenier. 2005. A Framework for annotating information structure in discourse. In *Frontiers in Corpus Annotation II: Pie in the Sky. Proceedings of the Workshop, ACL 2005*, Ann Arbor, MI, pp. 45–52.
- Carletta, J. 1996. Assessing agreement on classification tasks: The kappa statistics. *Computational Linguistics* 22(2): 249–254.
- Carroll, J. 1998. Evaluation: Parsing. *ELSNews: The Newsletter of the European Network in Language and Speech* 7(3): 8.
- Carroll, J., Minnen, G., and T. Briscoe. 2003. Parser evaluation with a grammatical relation annotation scheme. In *Treebanks: Building and Using Parsed Corpora*, ed. A. Abeillé, pp. 299–316. Dordrecht, the Netherlands: Kluwer.
- Cassidy, S. 2002. XQuery as an annotation query language: A use case analysis. In *Proceedings of LREC 2002*, Las Palmas, Canary Islands, Spain, pp. 2055–2060.
- Charniak, E. 1993. *Statistical Language Learning*. Cambridge, MA: MIT Press.
- Charniak, E. 1996. Tree-bank Grammars. In *Proceedings of the 13th National Conference on Artificial Intelligence*, Menlo Park, CA, pp. 1031–1036.

- Chen, J. and V. K. Shanker. 2004. Automated extraction of TAGs from the Penn Treebank. In *New Developments in Parsing Technology, Text, Speech And Language Technology*, Vol. 23, eds. H. Bunt, J. Carroll, and G. Satta, pp. 73–89. Norwell, MA: Kluwer Academic Publishers.
- Chen, K., Huang, C., Chen, F., Lao, C., Chang, M., and C. Chen. 2003. Sinica treebank: Design criteria, representational issues and implementation. In *Treebanks: Building and Using Parsed Corpora*, ed. A. Abeillé, pp. 231–248. Dordrecht, the Netherlands: Kluwer.
- Cieri, C. and S. Bird. 2001. Annotation graphs and servers and multi-modal resources: Infrastructure for interdisciplinary education, research and development. In *Proceedings of the ACL 2001 Workshop*

- on *Sharing Tools and Resources*, Toulouse, France, Vol. 15, pp. 23–30, July 07, 2001. *Annual Meeting of the ACL*. Association for Computational Linguistics, Morristown, NJ.
- Civit, M. and M. A. Martí. 2002. Design principles for a Spanish treebank. In *Proceedings of TLT 2002*, Sozopol, Bulgaria, eds. E. Hinrichs and K. Simov, pp. 61–77.
- Clark J. and S. DeRose. 1999. XML path language (XPath). <http://www.w3.org/TR/xpath>.
- Cohen, J. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement* 20(1): 37–46.
- Collins, M. 1997. Three generative, lexicalized models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, Somerset, NJ, eds. P. R. Cohen and W. Wahlster, pp. 16–23.
- Craggs, R. and M. M. Wood. 2005. Evaluating discourse and dialogue coding schemes. *Computational Linguistics* 31(3): 289–296.
- Daum, M., Foth, K., and W. Menzel. 2004. Automatic transformation of phrase treebanks to dependency trees. In *Proceedings of LREC 2004*, Lisbon, Portugal, pp. 1149–1152.
- Delmonte, R., Bristot, A., and S. Tonelli. 2007. VIT – Venice Italian treebank: Syntactic and quantitative features. In *Proceedings of TLT 2007*, Bergen, Norway, pp. 43–54.
- Dickinson, M. 2006a. Rule equivalence for error detection. In *Proceedings of TLT 2006*, Prague, Czech Republic, pp. 187–198.
- Dickinson, M. 2006b. From detecting errors to automatically correcting them. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06)*, Trento, Italy, pp. 265–272.
- Eugenio B. Di and M. Glass. 2004. The kappa statistic: A second look. *Computational Linguistics* 30(1): 95–101.
- Fellbaum, C. (ed.). 1998. *WordNet: An Electronic Lexical Database*. Cambridge, MA: MIT Press.
- Fillmore, C. J., Baker, C. F., and H. Sato. 2002. Seeing arguments through transparent structures. In *Proceedings of LREC 2002*, Las Palmas, Canary Islands, Spain, pp. 787–791.
- Fillmore, Ch. J., Johnson, Ch. R., and M. R. L. Petruck. 2003. Background to Framenet. *International Journal of Lexicography* 16(3): 235–250.
- Fligelstone, S. 1992. Developing a scheme for annotating text to show anaphoric relations. In *New Directions in English Language Corpora*, ed. G. Leitner, pp. 53–170. Berlin, Germany: Mouton de Gruyter.
- Grishman, R. and B. Sundheim. 1996. Design of the MUC-6 evaluation. In *Annual Meeting of the ACL—Proceedings of a Workshop on Held at Vienna*, Vienna, VA, pp. 413–422.
- Hajič, J. 1998. Building a syntactically annotated corpus: The Prague dependency treebank. In *Issues of Valency and Meaning. Studies in Honour of Jarmila Panevová*, ed. E. Hajičová, pp. 106–132. Prague, Czech Republic: Karolinum, Charles University Press.
- Hajič, J. and B. Hladká. 1998. Tagging inflective languages: Prediction of morphological categories for a rich, structured tagset. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, Montreal, QC, pp. 483–490. Montreal, QC: Association for Computational Linguistics.
- Hajičová, E. 2002. Theoretical description of language as a basis of corpus annotation: The case of Prague dependency treebank. *Prague Linguistic Circle Papers* 4: 111–127.

Hajičová, E. 2003. Topic-focus articulation in the Czech national corpus. In *Language and Function: To the Memory of Jan Firbas*, ed. J. Hladký, pp. 185–194. Amsterdam, the Netherlands: John Benjamins. Hajičová, E. and P. Sgall. 2001. Topic-focus and salience. In *Proceedings of 39th Annual Meeting of the Association for Computational linguistics*, Toulouse, France, pp. 276–281. Toulouse, France:

Association for Computational Linguistics.

Hajičová, E. and B. V. Hladká. 2008. What does sentence annotation say about discourse? In *Proceedings of the 18th International Congress of Linguists*, Seoul, South Korea, Vol. 2, pp. 125–126. Seoul, South Korea: The Linguistic Society of Korea.

- Hinrichs, E. W., Bartels, J., Kawata, Y., Kordoni, V., and H. Telljohann. 2000. The Tuebingen treebanks for spoken German, English, and Japanese. In *VerbMobil: Foundations of Speech-to-Speech Translation*, ed. W. Wahlster, pp. 550–574. Berlin, Germany: Springer-Verlag.
- Hockenmeier, J. and M. Steedman. 2002. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of LREC 2002*, Las Palmas, Spain, pp. 1974–1981.
- Ide, N., Bonhomme, P., and L. Romary. 2000. XCES: An XML-based standard for linguistic corpora. In *Proceedings of LREC 2000*, Athens, Greece, pp. 825–830.
- Jäborg, J. 1986. *SynTag Dokumentation. Manual för syntagging*. Göteborgs University, Institute för språkvetenskaplig databehandling, Gothenburg: Sweden.
- Johansson, S. 1980. The LOB corpus of British English texts: Presentation and comments. *ALLC Journal* 1(1): 25–36.
- Kallmeyer, L. 2000. On the complexity of queries for structurally annotated linguistic data. In *Proceedings of ACIDCA'2000*, pp. 105–110. Tunisia.
- Kawata, Y. and J. Barteles. 2000. *Stylebook for Japanese Treebank in VerbMobil*. Technical Report 240, VerbMobil, Eberhard-Karls-Universität, Tübingen, Germany.
- Kepser, S. 2003. Finite structure query — A tool for querying syntactically annotated corpora. In *Proceedings of EACL 2003*, Budapest, Hungary, pp. 179–186.
- Kingsbury, P. and M. Palmer. 2002. From TreeBank to PropBank. In *Proceedings of LREC 2002*, Las Palmas, Canary Islands, Spain, pp. 1989–1993.
- Krippendorff, K. 1980. *Content Analysis: An Introduction to Its Methodology*, Chapter 12. Beverly Hills, CA: Sage.
- Krippendorff, K. 2004. Reliability in content analysis: Some common misconceptions and recommendations. *Human Communication Research* 30(3): 411–433.
- Kromann, M. T. 2003. The Danish dependency treebank and the DTAG treebank tool. In *Proceedings of TLT 2003*, Växjö, Sweden, pp. 217–220.
- Kucera, H. and W. Francis. 1967. *Computational Analysis of Present Day American English*. Providence, RI: Brown University Press.
- Kurohashi, S. and M. Nagao. 2003. Building a Japanese parsed corpus while improving the parsing system. In *Treebanks: Building and Using Parsed Corpora*, ed. A. Abeillé, pp. 249–260. Dordrecht, the Netherlands: Kluwer.
- Lai, C. and S. Bird. 2004. Querying and updating treebanks: A critical survey and requirements analysis. In *Proceedings of the Australasian Language Technology Workshop*, Sydney, NSW, pp. 139–146.
- Landes, S., Leacock C., and R.I. Teng. 1998. Building semantic concordances. In *WordNet: An Electronic Lexical Database*, ed. C. Fellbaum, Cambridge, MA: MIT Press.
- Leech, G. and G. Roger. 1991. Running a grammar factory, the production of syntactically analyzed corpora or “treebanks”. In *English Computer Corpora*, pp. 15–32. Mouton de Gruyter, Berlin, Germany.
- Levin, B. 1993. *English Verb Classes and Alternations*. Chicago IL London, U.K.: University of Chicago Press.
- Lezius, W. 2002. *Ein Suchwerkzeug für syntaktisch annotierte Textkorpora*. PhD thesis IMS, University of Stuttgart, Stuttgart, Germany.
- Malrieu, D. and F. Rastier. 2001. Genres et variations morpho-syntaxiques. *Traitement automatique des langues: linguistique de corpus*, ed. Daille B. and Romary R., 42(2): 547–577.

- Marciniak, M., Mykowiecka, A., Przepiorkowski, A., and A. Kupsc. 2003. An HPSG-annotated test suite for polish. *Building and Using Parsed Corpora*, ed. A. Abeillé, pp. 129–146. Dordrecht, the Netherlands: Kluwer.
- Marcus, M., Santorini, B., and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics* 19(2): 313–330.
- Marcus, M., Kim, G., Marcinkiewicz, M. A. et al. 1994. The Penn treebank: Annotating predicate argument structure. In *Proceedings of the Human Language Technology Workshop*, Princeton, NJ, pp. 114–119. Princeton, NJ: Morgan Kaufmann Publishers Inc.

- McDonald, R. and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics: EACL 2006*, Trento, Italy, pp. 81–88.
- Merlo, P. and S. Stevenson. 2001. Automatic verb classification based on statistical distribution of argument structure, *Computational Linguistics* 27(3): 373–408.
- Meyers, A., Reeves, R., Macleod, C. et al. 2004. Annotating noun argument structure for NomBank. In *Proceedings of LREC 2004*, Lisbon, Portugal, pp. 803–806.
- Miltsakaki, E., Robaldo, L., Lee, A., and A. Joshi. 2008. Sense annotation in the Penn discourse Treebank, *Computational Linguistics and Intelligent Text Processing. Lecture Notes in Computer Science* 4919: 275–286.
- Mírovský, J. 2008a. PDT 2.0 requirements on a query language. In *Proceedings of ACL 2008*, Columbus, OH, pp. 37–45.
- Mírovský, J. 2008b. Netgraph—Making searching in treebanks easy. In *Proceedings of the Third International Joint Conference on Natural Language Processing (IJCNLP 2008)*, Hyderabad, India, pp. 945–950, January 8–10, 2008.
- Mladová, L., Zikánová, Š., and E. Hajičová. 2008. From sentence to discourse: Building an annotation scheme for discourse based on Prague Dependency Treebank. In *Proceedings of LREC 2008*, Marrakech, Morocco, pp. 1–7.
- Monachesi, P., Stevens, G., and J. Trapman. 2007. Adding semantic role annotation to a corpus of written Dutch. In *Proceedings of the Linguistic Annotation Workshop (LAW-07)*, Prague, Czech Republic, pp. 77–84. Stroudsburg, PA: Association for Computational Linguistics.
- Moreno, A., Lopez, S., and F. Sanchez. 2003. Developing a syntactic annotation scheme and tools for a Spanish Treebank. In *Treebanks: Building and Using Parsed Corpora*, ed. A. Abeillé, pp. 149–164. Dordrecht, the Netherlands: Kluwer.
- Nelson, G., Wallis, S., and B. Aarts. 2001. *Exploring Natural Language: Working with the British Component of the International Corpus of English*. Amsterdam, the Netherlands: J. Benjamins.
- Nivre, J. 2002. What kinds of trees grow in Swedish soil? A comparison of four annotation schemes for Swedish. In *Proceedings of TLT2002*, Sozopol, Bulgaria, eds. Hinrichs, E. and K. Simov, pp. 123–138. Nivre, J. 2003. Theory-supporting treebanks. In *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT 2003)*, Växjö University Press, Växjö, Sweden, ed. J. Nivre and E. Hinrichs, pp. 117–128.
- Nivre, J. 2006. *Inductive Dependency Parsing*. Text, Speech, and Language Technology Series, eds. N. Ide and J. Véronis, Dordrecht, the Netherlands: Springer, Vol. 34, p. 216, ISBN 1-4020-4888-2.
- Oepen, S., Flickinger, D., Toutanova, K., and Ch. D. Manning. 2002a. LinGO Redwoods: A rich and dynamic treebank for HPSG. In *Proceedings of TLT2002*, Sozopol, Bulgaria, pp. 139–149.
- Oepen, S., Toutanova, K., Shieber, S., Manning, Ch., Flickinger, D., and T. Brants. 2002b. The LinGO Redwoods treebank: Motivation and preliminary applications. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*, Taipei, Taiwan, pp. 1253–1257.
- Oflazer, K., Bilge, S., Hakkani-Tür, D. Z., and T. Gökhan. 2003. Building a Turkish treebank. In *Treebanks: Building and Using Parsed Corpora*, ed. A. Abeillé, pp. 261–277. Dordrecht, the Netherlands: Kluwer.
- Pajas, P. 2007. TrEd User's manual. <http://ufal.mff.cuni.cz/~pajas/tred>.
- Pajas, P. and J. Štěpánek. 2008. Recent advances in a feature-rich framework for treebank annotation. In *The 22nd International Conference on Computational Linguistics—Proceedings of the Conference*, pp. 673–680. The Coling 2008 Organizing Committee, Manchester, U.K., ISBN 978-1-905593-45-3.

- Palmer, M., Kingsbury, P., and D. Gildea. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics* 31(1):71–106.
- Pitler, E., Raghupathy, M., Mehta, H., Nenkova, A., Lee, A., and A. Joshi. 2008. Easily identifiable discourse relations. In *Proceedings of COLING 2008: Companion Volume: Posters and Demonstrations*, Manchester, U.K.
- Pito, R. 1994. TGrep Manual Page. <http://www ldc.upenn.edu/ldc/online/treebank>.

- Prasad, R., Dinesh, N., Lee A. et al. 2008. The Penn Discourse Treebank 2.0. In *Proceedings of LREC 2008*, Marrakech, Morocco, pp. 2961–2968.
- Prasad, R., Dinesh, N., Lee, A., Joshi, A., and B. Webber. 2007. Attribution and its Annotation in the Penn Discourse TreeBank. *Traitement Automatique des Langues, Special Issue on Computational Approaches to Document and Discourse* 47(2): 43–64.
- Prokopidis, P., Desypri, E., Koutsombogera, M., Papageorgiou, H., and S. Piperidin. 2005. Theoretical and practical issues in the construction of a Greek dependency treebank. In *Proceedings of TLT 2005*, Universitat de Barcelona, Barcelona, Spain, eds. M. Civit, S. Kübler, and M. A. Martí, pp. 149–160.
- Pynte, J. and S. Colonna. 2000. Decoupling syntactic parsing from visual inspection: The case of relative clause attachment in French. In *Reading as a Perceptual Process*, eds. A. Kennedy, R. Radach, D. Heller, and J. Pynte, pp. 529–547. Oxford, U.K.: Elsevier.
- Rohde, D. 2005. TGrep2 user manual. <http://www-cgi.cs.cmu.edu/~dr/TGrep2/tgrep2.pdf>.
- Rosén, V., De Smedt, K., and P. Meurer. 2006. Towards a toolkit linking treebanking and grammar development. In *Proceedings of TLT 2006*, Prague, Czech Republic, pp. 55–66.
- Roventini, A., Olivieri, M., and N. Calzolari. 2002. Integrating two semantic lexicons, SIMPLE and ItalWordNet: What can we gain? In *Proceedings of LREC 2002*, Las Palmas, Canary Islands, Spain, pp. 1473–1477.
- Sampson, G. 1995. *English for the Computer*. Oxford, U.K.: Oxford University Press.
- Scott, W. A. 1955. Reliability of content analysis: The case of nominal scale coding. *Public Opinion Quarterly* 19(3), 321–325.
- Ševčíková, M., Žabokrtský, Z., and O. Krůza. 2007. Named entities in Czech: Annotating data and developing NE tagger. In *Proceedings of the 10th International Conference on Text, Speech and Dialogue*, Pilsen, Czech Republic, pp. 188–195. *Lecture Notes In Computer Science*. Pilsen, Czech Republic: Springer.
- Simov, K. 2001. Grammar extraction from an HPSG corpus. In *Proceedings of the RANLP 2001 Conference*, Tzigov Chark, Bulgaria, pp. 285–287.
- Simov, K., Osenova, P., Slavcheva, M. et al. 2002. Building a linguistically interpreted corpus for Bulgarian: The Bultreebank project. In *Proceedings of LREC 2002*, Las Palmas, Canary Islands, Spain, pp. 1729–1736.
- Simov, K., Popova, G., and P. Osenova. 2001. HPSG-based syntactic treebank of Bulgarian (BulTreeBank). In *Proceedings of the Corpus Linguistics 2001 Conference*, Lancaster, U.K., p. 561.
- Slavcheva, M. 2002. Segmentation layers in the group of the predicate: A case study of Bulgarian within the BulTreeBank framework. In *Proceedings of TLT 2002*, Sozopol, Bulgaria, pp. 199–210.
- Srinivas, B. and A. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics* 25(2): 237–266.
- Steiner, I. and L. Kallmeyer. 2002. VIQTORYA — A visual tool for syntactically annotated corpora. In *Proceedings of LREC 2002*, Las Palmas, Canary Islands, Spain, pp. 1704–1711.
- Surdeanu, M., Johansson, R., Meyers, A., Marquez, L., and J. Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the 12th Conference on Computational Natural Language Learning (CoNLL-2008)*, Manchester, U.K.
- Toutanova, K. and Ch. D. Manning. 2002. Feature selection for a rich HPSG grammar using decision trees. In *Proceedings of the Sixth Conference on Natural Language Learning (CoNLL 2002)*, Taipei, Taiwan, pp. 1–7.
- Toutanova, K., Manning, Ch. D., and S. Oepen. 2002. Parse disambiguation for a rich HPSG grammar. In *Proceedings of TLT 2002*, Sozopol, Bulgaria, pp. 253–263.

- Tutin, A., Trouilleux, F., Clouzot, C., and E. Gaussier. 2000. Building a large corpus with anaphoric links in French: Some methodological issues. In *Actes de Discourse Anaphora and Reference Resolution Colloquium*, Lancaster, U.K.
- Ustkoreit, H. 2004. New chances for deep linguistic processing. In *Proceedings of 19th International Conference on Computational Linguistics: COLING-2002*, Taipei, Taiwan, pp. 15–27.

van der Beek, L., Bouma, G., Malouf, R., and G. van Noord. 2001. The Alpino dependency treebank. In *Computational Linguistics in the Netherlands CLIN 2001*, pp. 8–22. Amsterdam, the Netherlands: Rodopi.

Velldal, E., Oepen, S., and D. Flickinger. 2004. Paraphrasing treebanks for stochastic realization ranking. In *Proceedings of TLT 2004*, Tuebingen, Germany, pp. 149–160.

Véronis, J. and L. Khouri. 1995. Étiquetage grammatical multilingue: le projet MULTTEXT. *TAL* 36(1–2): 233–248.

Veselá, K., Havelka, J., and E. Hajičová. 2004. Annotators’ agreement: The case of topic-focus articulation. In *Proceedings of LREC 2004*, Lisbon, Portugal, pp. 2191–2194.

Wallis, S. 2003. Completing parsed corpora: From correction to evolution. In *Treebanks: Building and Using Parsed Corpora*, ed. A. Abeillé, pp. 61–71. Dordrecht, the Netherlands: Kluwer.

Webber, B. and A. Joshi. 1998. Anchoring a lexicalized tree-adjoining grammar for discourse. In *Proceedings of ACL/COLING Workshop on Discourse Relations and Discourse Markers*, Montreal, QC, pp. 86–92.

Xia, F., Palmer, M., Xue, N. et al. 2000. Developing guidelines and ensuring consistency for Chinese text annotation. In *Proceedings of LREC 2000*, Athens, Greece, pp. 3–10.

Žabokrtský, Z. and I. Kučerová. 2002. Transforming Penn treebank phrase trees into (Praguan) tectogrammatical dependency trees *The Prague Bulletin of Mathematical Linguistics* 78:77–94.

Fundamental Statistical Techniques

9.1	Binary Linear Classification	189
9.2	One-versus-All Method for Multi-Category Classification	193
9.3	Maximum Likelihood Estimation	194
9.4	Generative and Discriminative Models	194
	Naive Bayes · Logistic Regression	
9.5	Mixture Model and EM	198
9.6	Sequence Prediction Models	200
	Hidden Markov Model · Local Discriminative Model for Sequence Prediction · Global Discriminative Model for Sequence Prediction	
	References	204

Tong Zhang
Rutgers, The State
University of New Jersey

The statistical approach to natural language processing (NLP) has become more and more important in recent years. This chapter gives an overview of some fundamental statistical techniques that have been widely used in different NLP tasks. Methods for statistical NLP mainly come from machine learning, which is a scientific discipline concerned with learning from data. That is, to extract information, discover patterns, predict missing information based on observed information, or more generally construct probabilistic models of the data. Machine learning techniques covered in this chapter can be divided into two types: supervised and unsupervised.

Supervised learning is mainly concerned with predicting missing information based on observed information. For example, predicting part of speech (POS) based on sentences. It employs statistical methods to construct a prediction rule from labeled training data. Supervised learning algorithms discussed in this chapter include naive Bayes, support vector machines (SVMs), and logistic regression. The goal of unsupervised learning is to group data into clusters. The main statistical techniques are mixture models and the expectation maximization (EM) algorithm. This chapter will also cover methods used in sequence analysis, such as hidden Markov model (HMM), conditional random field (CRF), and the Viterbi decoding algorithm.

9.1 Binary Linear Classification

The goal of binary classification is to predict an unobserved binary label $y \in \{-1, 1\}$, based on an observed input vector $x \in R^d$. A classifier $h(x)$ maps $x \in R^d$ to $\{-1, 1\}$. If it agrees with the label y , the error is zero. If it does not agree with the label y , we suffer a loss of one. That is, the classification error is defined as

189

$$\text{err}(h(x), y) = \begin{cases} 1 & h(x) \neq y \\ 0 & \text{otherwise,} \end{cases}$$

where $I(\cdot)$ is the set indicator function.

A commonly used method for binary classification is to learn a real-valued scoring function $f(x): R^d \rightarrow R$ that induces a classification rule

$$h(x) = \begin{cases} 1 & f(x) > 0 \\ -1 & \text{otherwise.} \end{cases} \quad (9.1)$$

A commonly used scoring function is linear: $f(x) = w^T x + b$, where $w \in R^d$ and $b \in R$. A binary classification method using linear scoring function is called a linear classifier.

In supervised learning, the classifier $h(x)$, or its scoring function $f(x)$, is learned from a set of labeled examples

$$\{(x_1, y_1), \dots, (x_n, y_n)\},$$

referred to as training data. Its performance (average classification error) should be evaluated on a separate set of labeled data called test data.

A procedure that constructs a scoring function $f(x)$ from the training data is called a learning algorithm. For example, a standard learning algorithm for linear classification is linear least squares method, which finds a weight vector $\hat{w} \in R^d$ and bias $\hat{b} \in R$ for a linear scoring function $f(x) = \hat{w}^T x + \hat{b}$ by minimizing the squared error on the training set:

$$[\hat{w}, \hat{b}] = \arg \min_{w, b} \sum_{i=1}^n (w^T x_i + b - y_i)^2. \quad (9.2)$$

Using linear algebra, we may write the solution of this formula in closed form as

$$\hat{w} = \left(\sum_{i=1}^n (x_i - \bar{x})(x_i - \bar{x})^T \right)^{-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}),$$

$$\hat{b} = \bar{y} - \hat{w}^T \bar{x},$$

where

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad \text{and} \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i.$$

One problem of the above formulation is that the matrix $\sum_{i=1}^n (x_i - \bar{x})(x_i - \bar{x})^T$ may be singular or ill-conditioned (this occurs, e.g., when n is small and the data points are not well distributed). A standard remedy is to use the ridge regression method (Hoerl and Kennard 1970) that adds a regularization term $\lambda w^T w$ to (9.2). For

convenience, we set $b = 0$:

$$\hat{w} = \arg \min_w \frac{\sum_{i=1}^n (w^T x_i y_i - 1)^2}{n} + \lambda w^T w, \quad (9.3)$$

where $\lambda > 0$ is an appropriately chosen regularization parameter. The solution is given by

$$\hat{w} = \left(\sum_{i=1}^n x_i x_i^T + \lambda n I \right)^{-1} \sum_{i=1}^n x_i y_i,$$

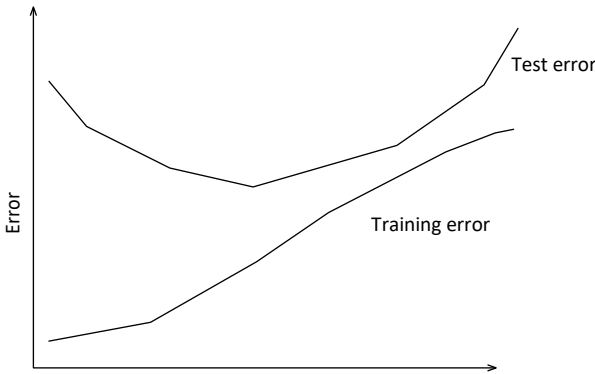
where I denotes the identity matrix. This method solves the ill-conditioning problem because $\sum_{i=1}^n x_i x_i^T + \lambda nI$ is always non-singular.

Note that taking $b = 0$ in (9.3) does not make the resulting scoring function $f(x) = \hat{w}^T x$ less general. To see this, one can embed all the data into a space with one more dimension with some constant A (normally, one takes $A = 1$). In this conversion, each vector $x_i = [x_{i,1}, \dots, x_{i,d}]$ in the original space becomes the vector $x^j = [x_{i,1}, \dots, x_{i,d}, A]$ in the larger space. Therefore, the linear classifier $w^T x + b = w^j T x^j$, where $w^j = [w, b]$ is a weight vector in $(d + 1)$ -dimensional space. Due to this simple change of representation, the linear scoring function with b in the original space is equivalent to a linear scoring function without b in the larger space.

The introduction of the regularization term $\lambda w^T w$ in (9.3) makes the solution more stable. That is, a small perturbation of the observation does not significantly change the solution. This is a desirable property because the observations (both x_i and y_i) often contain noise. However, λ introduces a bias into the system because it pulls the solution $\hat{w}$ toward zero. When $\lambda \rightarrow \infty$, $\hat{w} \rightarrow 0$. Therefore, it is necessary to balance the desirable stabilization effect and the undesirable bias effect, so that the optimal trade-off can be achieved. Figure 9.1 illustrates the training error versus test error when λ changes. As λ increases, due to the bias effect, the training error always increases. However, since the solution becomes more robust to noise as λ increases, the test error will decrease first. This is because the benefit of a more stable solution is larger than the bias effect. After the optimal trade-off (the lowest test error point) is achieved, the test error becomes larger when λ increases. This is because the benefit of more stability is smaller than the increased bias.

In practice, the optimal λ can be selected using cross-validation, where we randomly split the training data into two parts: a training part and a validation part. We use only the first (training) part to compute $\hat{w}$ with different λ , and then estimate its performance on the validation part. The λ with the smallest validation error is then chosen as the optimal regularization parameter.

The decision rule (9.1) for a linear classifier $f(x) = w^T x + b$ is defined by a decision boundary $\{x : w^T x + b = 0\}$: on one side of this hyperplane, we predict $h(x) = 1$, and on the other side, we predict $h(x) = -1$. If the hyperplane completely separates the positive data from the negative data without error, we call it a separating hyperplane. If the data are linearly separable, then there can be more than one possible separating hyperplanes, as shown in Figure 9.2. A natural question is: What is a better separating hyperplane? One possible measure to define the quality of a separating hyperplane is through the concept of margin, which is the distance of the nearest training example to the linear decision boundary. A separating hyperplane with a larger margin is more robust to noise because training data can still be separated after a small perturbation. In Figure 9.2, the boundary represented by the solid line has a larger margin than the boundary represented by the dashed line, and thus it is the preferred classifier.



Lambda

FIGURE 9.1 Effect of regularization.

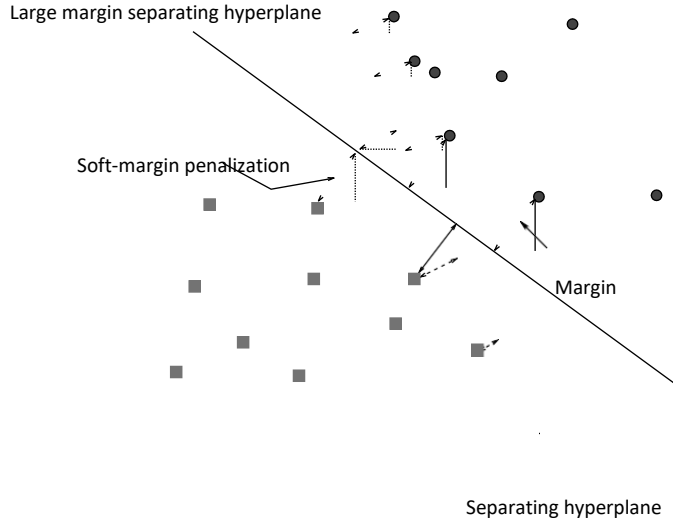


FIGURE 9.2 Margin and linear separating hyperplane.

The idea of finding an optimal separating hyperplane with largest margin leads to another popular linear classification method called support vector machine (Cortes and Vapnik 1995; Joachims 1998). If the training data are linearly separable, the method finds a separating hyperplane with largest margin defined as

$$\min_{i=1, \dots, n} (w^T x_i + b) y_i / \|w\|_2.$$

The equivalent formulation is to minimize $\|w\|_2^2$ under the constraint $\min_i (w^T x_i + b) y_i \geq 1$. That is, the optimal hyperplane is the solution to

$$\begin{aligned} [\hat{w}, \hat{b}] &= \arg \min_{w, b} \|w\|_2^2 \\ \text{subject to } (w^T x_i + b) y_i &\geq 1 \quad (i = 1, \dots, n). \end{aligned}$$

For training data that is not linearly separable, the idea of margin maximization cannot be directly applied. Instead, one considers the so-called soft-margin formulation as follows:

$$\begin{aligned} [\hat{w}, \hat{b}] &= \arg \min_{w, b} \|w\|_2^2 + C \sum_{i=1}^n \xi_i, \\ \text{subject to } y_i (w^T x_i + b) &\geq 1 - \xi_i, \quad \xi_i \geq 0, \quad (i = 1, \dots, n). \end{aligned} \quad (9.4)$$

In this method, we do not require that all training data can be separated with margin at least one. Instead, we introduce soft-margin slack variable $\xi_i \geq 0$ that penalize points with smaller margins. The parameter $C \geq 0$ balances the margin violation (when $\xi_i > 0$) and the regularization term $\|w\|_2^2$. When $C \rightarrow \infty$, we have $\xi_i \rightarrow 0$; therefore, the margin condition $y_i (w^T x_i + b) \geq 1$ is enforced for all i . The resulting method

becomes equivalent to the separable SVM formulation.

By eliminating ξ_i from (9.4), and let $\lambda = 1/(nC)$, we obtain the following equivalent formulation:

$$[\hat{w}, \hat{b}] = \arg \min_{w,b} \sum_{i=1}^n \frac{1}{n} g((w^\top x_i + b)y_i) + \lambda \|w\|_2^2, \tag{9.5}$$

where

$$g(z) = \begin{cases} 1 - z & \text{if } z \leq 1, \\ 0 & \text{if } z > 0. \end{cases} \tag{9.6}$$

This method is rather similar to the ridge regression method (9.3). The main difference is a different loss function $g(\cdot)$, which is called hinge loss in the literature. Compared to the least squares loss, the hinge loss does not penalize data points with large margin.

9.2 One-versus-All Method for Multi-Category Classification

In practice, one often encounters multi-category classification, where the goal is to predict a label $y \in \{1, \dots, k\}$ based on an observed input x .

If we have a binary classification algorithm that can learn a scoring function $f(x)$ from training data $\{(x_i, y_i)\}_{i=1, \dots, n}$ with $y_i \in \{-1, 1\}$, then it can also be used for multi-category classification.

A commonly used method is one-versus-all. Consider a multi-category classification problem with k classes: $y_i \in \{1, \dots, k\}$. We may reduce it into k binary classification problems indexed by class label $4 \in \{1, \dots, k\}$. The 4 th problem has training data $(x, y^{(4)})$ ($i = 1, \dots, n$), where we define the binary

i

label $y^{(4)} \in \{-1, +1\}$ as

$$y^{(4)} = \begin{cases} 1 & \text{if } y_i = 4, \\ -1 & \text{otherwise.} \end{cases}$$

For each binary problem 4 defined this way with training data $\{(x, y^{(4)})\}$, we may use a binary classification algorithm to learn a scoring function $f_4(x)$. For example, using linear SVM or linear least squares, we can learn a linear scoring function of the form $f_4(x) = w^{(4)}x + b^{(4)}$ for each 4 . For a data point x , the higher the score $f_4(x)$, the more likely x belongs to class 4 . Therefore, the classification rule for the multi-class problem is

$$h(x) = \arg \max_{4 \in \{1, \dots, k\}} f_4(x).$$

Figure 9.3 shows the decision boundary for three classes with linear scoring functions $f_4(x)$ ($4 = 1, 2, 3$). The three dashed lines represent the decision boundary $f_4(x) = 0$ ($4 = 1, 2, 3$) for the three binary problems. The three solid lines represent the decision boundary of the multi-class problem, determined by the lines $f_1(x) = f_2(x)$, $f_1(x) = f_3(x)$, and $f_2(x) = f_3(x)$, respectively.

o

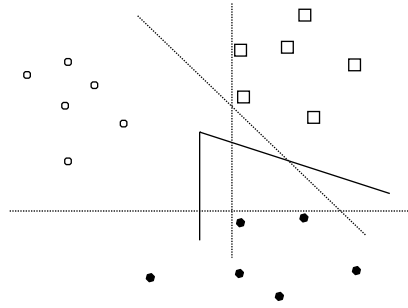


FIGURE 9.3 Multi-class linear classifier decision boundary.

9.3 Maximum Likelihood Estimation

A very general approach to machine learning is to construct a probability model of each individual data point as $p(x, y|\theta)$, where θ is the model parameter that needs to be estimated from the data. If the training data are independent, then the probability of the training data is

$$\prod_{i=1}^n p(x_i, y_i|\theta).$$

A commonly used statistical technique for parameter estimation is the maximum likelihood method, which finds a parameter $\hat{\theta}$ by maximizing the likelihood of the data $\{(x_1, y_1), \dots, (x_n, y_n)\}$:

$$\hat{\theta} = \arg \max_{\theta} \prod_{i=1}^n p(x_i, y_i|\theta).$$

More generally, we may impose a prior $p(\theta)$ on θ , and use the penalized maximum likelihood as follows:

$$\hat{\theta} = \arg \max_{\theta} \sum p(\theta) \prod_{i=1}^n p(x_i, y_i|\theta).$$

In the Bayesian statistics literature, this method is also called MAP (maximum a posterior) estimator. A more common way to write the estimator is to take logarithm of the right-hand side:

$$\hat{\theta} = \arg \max_{\theta} \sum_{i=1}^n \ln p(x_i, y_i|\theta) + \ln p(\theta).$$

For multi-classification problems with k classes $\{1, \dots, k\}$, we obtain the following class conditional probability estimate:

$$p(y|x) = \frac{p(x, y|\theta)}{\sum_{k=1}^k p(x, k|\theta)} \quad (y = 1, \dots, k).$$

The class conditional probability function may be regarded as a scoring function, with the following classification rule that chooses the class with the largest conditional probability:

$$h(x) = \arg \max_{y \in \{1, \dots, k\}} p(y|x).$$

9.4 Generative and Discriminative Models

We shall give two concrete examples of maximum likelihood estimation for supervised learning. In the

literature, there are two types of probability models called generative model and discriminative model. In a generative model, we model the conditional probability of input x given the label y ; in a discriminative model, we directly model the conditional probability $p(y|x)$. This section describes two methods: naive Bayes, a generative model, and logistic regression, a discriminative model. Both are commonly used linear classification methods.

9.4.1 Naive Bayes

The naive Bayes method starts with a generative model as in (9.7). Let $\theta = \{\theta^{(4)}\}_{4=1, \dots, k}$ be the model parameter, where we use a different parameter $\theta^{(4)}$ for each class 4. Then we can model the data as

$$p(x, y | \theta) = p(y)p(x|y, \theta) \quad \text{and} \quad p(x|y, \theta) = p(x|\theta^{(y)}). \quad (9.7)$$

This probability model can be visually represented using a graphical model as in Figure 9.4, where the arrows indicate the conditional dependency structure among the variables.

The conditional class probability is

$$p(y|x) = \frac{p(y)p(x|\theta^{(y)})}{\sum_{4=1}^k p(y=4)p(x|\theta^{(4)})}.$$

In the following, we shall describe the multinomial naive Bayes model (McCallum and Nigam 1998) for $p(x|\theta^{(y)})$, which is important in many NLP problems. In this model, the observation x represents multiple (unordered) occurrences of d possible symbols. For example, x may represent the number of word occurrences in a text document by ignoring the word order information (such a representation is often referred to as “bag of words”). Each word in the document is one of d possible symbols from a dictionary.

Specifically, each data point x_i is a d -dimensional vector $x_i = [x_{i,1}, \dots, x_{i,d}]$ representing the number of occurrences of these d symbols: For each symbol j in the dictionary, $x_{i,j}$ is the number of occurrences of symbol j . For each class 4, we assume that words are independently drawn from the dictionary according to a probability distribution $\theta^{(4)} = [\theta_1^{(4)}, \dots, \theta_d^{(4)}]$: that is, the symbol j occurs with probability $\theta_j^{(4)}$.

Now, for a data point x_i with label $y_i = 4$, x_i comes from a multinomial distribution:

$$p(x_i | \theta^{(4)}) = \frac{\bar{\theta}^{(4)} x_i}{\prod_{j=1}^d (\theta_j^{(4)})^{x_{i,j}}} \prod_{j=1}^d p^{x_{i,j}},$$

where we make the assumption that the total number of occurrences $\sum_{j=1}^d x_{i,j}$ is independent of the label y_i as

$y_i = 4$. For each $y \in \{1, \dots, k\}$, we consider the so-called Dirichlet prior for θ

$$\bar{p}(\theta^{(y)}) \propto \prod_{j=1}^d (\theta_j^{(y)})^{\lambda}$$

where $\lambda > 0$ is a tuning parameter. We may use the MAP estimator to compute $\theta^{(4)}$ separately for each class 4:

$$\hat{\theta}^{(4)} = \arg \max_{\theta \in \mathbb{R}^d} \left[\sum_{i: y_i=4} \sum_{j=1}^d x_{i,j} \log \theta_j^{(4)} + \sum_{j=1}^d \lambda \log \theta_j^{(4)} \right]$$

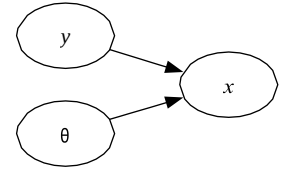


FIGURE 9.4 Graphical representation of a generative model.

$$(\theta_j)^{ij} \qquad \qquad \qquad \sum_{j=1}^d (\theta_j) \leq 1$$

$$\text{subject to} \qquad \theta_j = 1, \quad \text{and} \quad \theta_j \geq 0 \; (j = 1, \ldots, d).$$

$$\text{The solution is given by} \qquad \qquad \qquad \sum_{j=1}^d$$

$$\hat{\theta}_j^{(4)} = \frac{n^{(4)}}{\sum_{j=1}^d n_j^{(4)}},$$

where

$$n^{(4)} = \sum_j \sum_{i: y_i = 4} x_{ij}.$$

Let $n^{(4)} = \sum_{i: y_i = 4} 1$ be the number of training data with class label 4 for each $4 = 1, \dots, k$, then we may estimate

$$p(y) = n^{(y)}/n.$$

With the above estimates, we obtain a scoring function

$$f_4(x) = \ln p(x|\theta^{(y)}) + \ln p(y) = (\hat{w}^{(4)})^T x + \hat{b}^{(4)},$$

where

$$\hat{w}^{(4)} = [\ln \hat{\theta}_j^{(4)}]_{j=1, \dots, d} \quad \text{and} \quad \hat{b}^{(4)} = \ln(n^{(4)}/n).$$

The conditional class probability is given by the Bayes rule:

$$p(y|x) = \frac{p(x|y)p(y)}{\sum_{4=1}^k p(x|4)p(4)} = \frac{e^{f_y(x)}}{\sum_{4=1}^k e^{f_4(x)}} \quad (9.8)$$

and the corresponding classification rule is

$$h(x) = \arg \max_{4 \in \{1, \dots, k\}} f_4(x).$$

9.4.2 Logistic Regression

Naive Bayes is a generative model in which we model the conditional probability of input x given the label y . After estimating the model parameter, we may then obtain the desired class conditional probability $p(y|x)$ using the Bayes rule. A different approach is to directly model the conditional probability $p(y|x)$. Such a model is often called a discriminative model. The dependency structure is given by Figure 9.5.

Ridge regression can be interpreted as the MAP estimator for a discriminative model with Gaussian noise (note that although ridge regression can be applied to classification problems, the underlying Gaussian noise assumption is only suitable for real-valued output) and a Gaussian prior. The probability model is (with parameter $\theta = w$)

$$p(y|w, x) = N(w^T x, \tau^2),$$

with prior on parameter

$$p(w) = N(0, \sigma^2).$$

Here, we shall simply assume that σ^2 and τ^2 are known variance parameters, and the only unknown parameter is w . The MAP estimator is

$$\hat{w} = \arg \min_w \frac{1}{2} \sum_{i=1}^n \tau^2 (w^T x_i - y_i)^2 + \frac{1}{2} w^T w$$

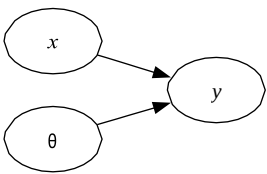


FIGURE 9.5 Graphical representation of a discriminative model.

which is equivalent to the ridge regression method in (9.3) with $\lambda = \tau^2/\sigma^2$.

However, for binary classification, since $y_i \in \{-1, 1\}$ is discrete, the noise $w^T x_i - y_i$ cannot be a Gaussian distribution. The standard remedy to this problem is logistic regression, which models the conditional class probability as

$$p(y = 1 | w, x) = \frac{1}{\exp(-w^T x) + 1}.$$

This means that both for $y = 1$ and $y = -1$, the likelihood is

$$p(y | w, x) = \frac{1}{\exp(-w^T x y) + 1}. \quad (9.9)$$

If we again assume a Gaussian prior $p(w) = N(0, \sigma^2)$, then the penalized maximum likelihood estimate is

$$\hat{w} = \arg \min_w \sum_{i=1}^n \ln(1 + \exp(-w^T x_i y_i)) + \lambda w^T w, \quad (9.10)$$

where $\lambda = 1/(2\sigma^2)$. Its use in text categorization as well as numerical algorithms for solving the problem can be found in Zhang and Oles (2001).

Although binary logistic regression can be used to solve multi-class problems using the one-versus-all method described earlier, there is a direct formulation of multi-class logistic regression, which we shall describe next. Assume we have k classes, the naive Bayes method induces a probability of the form (9.8), where each function $f_4(x)$ is linear. Therefore, as a direct generalization of the binary logistic model in (9.9), we may consider multi-category logistic model:

$$p(y | w^{(4)}, x) = \frac{e^{(w^{(y)})^T x}}{\sum_{4=1}^k e^{(w^{(4)})^T x}}. \quad (9.11)$$

The binary logistic model is a special case of (9.11) with $w^{(1)} = w$ and $w^{(-1)} = 0$. If we further assume Gaussian priors for each $w^{(4)}$,

$$P(w^{(4)}) = N(0, \sigma^2) \quad (4 = 1, \dots, k),$$

then we have the following MAP estimator:

$$\{\hat{w}^{(4)}\} = \arg \min_{\{w^{(4)}\}} \sum_{i=1}^n - (w^{(y_i)})^T x_i + \ln \frac{1}{k} e^{(w^{(4)})^T x_i} + \sum_{4=1}^k \frac{1}{k} (w^{(4)})^T w^{(4)},$$

λ

where $\lambda = 1/(2\sigma^2)$.

Multi-class logistic regression is also referred to as the maximum entropy method (MaxEnt) (Berger et al. 1996) under the following more general form:

$$P(y|w, x) = \frac{\exp(w^T z(x, y))}{\sum_{k=1}^K \exp(w^T z(x, y))}, \quad (9.12)$$

where $z(x, y)$ is a human-constructed vector called feature vector that depends both on x and on y . Let $w = [w^{(1)}, \dots, w^{(k)}] \in R^{kd}$, and $z(x, y) = [0, \dots, 0, x, 0, \dots, 0]$, where x appears only in the positions $(y-1)d+1$ to yd that corresponds to $w^{(y)}$. With this representation, we have $w^T z(x, y) = (w^{(y)})^T x$, and (9.11) becomes a special case of (9.12).

Although logistic regression and naive Bayes share the same conditional class probability model, a major advantage of the logistic regression method is that it does not make any assumption on how x is generated. In contrast, naive Bayes assumes that x is generated in a specific way, and uses such information to estimate the model parameters. The logistic regression approach shows that even without any assumptions on x , the conditional probability can still be reliably estimated using discriminative maximum likelihood estimation.

9.5 Mixture Model and EM

Clustering is a common unsupervised learning problem. Its goal is to group unlabeled data into clusters so that data in the same cluster are similar, while data in different clusters are dissimilar. In clustering, we only observe the input data vector x , but do not observe its cluster label y . Therefore, it is called unsupervised learning.

Clustering can also be viewed from a probabilistic modeling point of view. Assume that the data belong to k clusters. Each data point is a vector x_i , with $y_i \in \{1, \dots, k\}$ being its corresponding (unobserved) cluster label. Each y_i takes value $4 \in \{1, \dots, k\}$ with probability $p(y_i = 4|x_i)$.

The goal of clustering is to estimate $p(y_i = 4|x_i)$. Similar to the naive Bayes approach, we start with a generative model of the following form:

$$p(x|\theta, y) = p(x|\theta^{(y)}).$$

Since y is not observed, we integrate out y to obtain

$$p(x|\theta) = \sum_{c=1}^k \mu_4 p(x|\theta^{(4)}), \quad (9.13)$$

where $\mu_4 = p(y = 4) (4 = 1, \dots, k)$ are k parameters to be estimated from the data. The model in (9.13), with missing data (in this case, y) integrated out, is called a mixture model. A cluster $4 \in \{1, \dots, k\}$ is referred to as a mixture component.

We can interpret the data generation process in (9.13) as follows. First we pick a cluster 4 (mixture component) from $\{1, \dots, k\}$ with a fixed probability μ_4 as y_i ; then we generate data points x_i according to the probability distribution $p(x_i|\theta^{(4)})$.

In order to obtain the cluster conditional probability $p(y = 4|x)$, we can simply apply Bayes rule:

$$p(y|x) = \frac{\mu_y p(x|\theta^{(y)})}{\sum_{4=1}^k \mu_4 p(x|\theta^{(4)})}. \quad (9.14)$$

Next we show how to estimate the model parameters $\{\theta^{(4)}, \mu_4\}$ from the data. This can be achieved using the penalized maximum likelihood method:

$$\hat{\theta}^{(4)} = \arg \max_{\theta} \sum_{i=1}^n \log p(x_i|\theta^{(4)}) - \frac{\lambda}{2} \sum_{i=1}^n \theta_i^2$$

$$\hat{\theta}^{(4)}, \hat{\mu}^{(4)} = \arg \max_{\{\theta^{(4)}, \mu^{(4)}\}} \sum_{i=1}^n \ln \mu^{(4)} p(x_i | \theta^{(4)}) + \sum_{i=1}^n \ln p(\theta^{(4)}) . \quad (9.15)$$

A direct optimization of (9.15) is usually difficult because the sum over the mixture components k is inside the logarithm for each data point. However, for many simple models such as the naive Bayes example considered earlier, if we know the label y_i for each x_i , then the estimation becomes easier: we simply estimate the parameters using the equation

$$\hat{\theta}^{(4)}, \hat{\mu}^{(4)} = \arg \max_{\{\theta^{(4)}, \mu^{(4)}\}} \sum_{i=1}^n \ln \mu_{y_i}^{(4)} p(x_i | \theta^{(4)}) + \sum_{i=1}^n \ln p(\theta^{(4)}) ,$$

which does not have the sum inside the logarithm. For example, in the naive Bayes model, both μ_4 and $\theta^{(4)}$ can be estimated using simple counting.

The EM algorithm (Dempster et al. 1977) simplifies the mixture model estimation problem by removing the sum over 4 inside the logarithm in (9.15). Although we do not know the true value of y_i , we can estimate the conditional probability of $y_i = 4$ for $4 = 1, \dots, k$ using (9.14). This can then be used to move the sum over 4 inside the logarithm to a sum over 4 outside the logarithm: For each data point i , we weight each mixture component 4 by the estimated conditional class probability $p(y_i = 4|x_i)$. That is, we repeatedly solve the following optimization problem:

$$[\hat{\theta}_{new}^{(4)}, \hat{\mu}_4^{new}] = \arg \max_{\theta^{(4)}, \mu_4} \sum_{i=1}^n p(y_i = 4|x_i, \hat{\theta}_{old}^{(4)}, \hat{\mu}_4^{old}) \ln [\mu_4 p(x_i|\theta^{(4)})] + \ln p(\theta^{(4)}) \quad (4)$$

for $4 = 1, \dots, k$. Each time, we start with $[\hat{\theta}_{old}^{(4)}, \hat{\mu}_4^{old}]$ and update its value to $[\hat{\theta}_{new}^{(4)}, \hat{\mu}_4^{new}]$. Note that the solution of $\hat{\mu}_4^{new}$ is

$$\hat{\mu}_4^{new} = \frac{\sum_{i=1}^n p(y_i = 4|x_i, \hat{\theta}_{old}^{(4)}, \hat{\mu}_4^{old}) x_i}{\sum_{i=1}^n p(y_i = 4|x_i, \hat{\theta}_{old}^{(4)}, \hat{\mu}_4^{old})}.$$

The algorithmic description of EM is given in Figure 9.6.

In practice, a few dozen iterations of EM often gives a satisfactory result. It is also necessary to start EM with different random initial parameters. This is to improve local optimal solutions found by the algorithm with each specific initial parameter configuration.

The EM algorithm can be used with any generative probability model including the naive Bayes model discussed earlier. Another commonly used model is Gaussian, where we assume $p(x|\theta^{(4)}) \propto \exp[-\frac{(\theta^{(4)}x)^2}{2\sigma^2}]$. Figure 9.7 shows a two-dimensional Gaussian mixture model with two mixture components represented by the dotted circles. For simplicity, we may assume that σ^2 is known. Under this assumption, Figure 9.6 can be used to compute the mean vectors $\theta^{(4)}$ for the Gaussian mixture model, where the E and M steps are given by

$$\begin{aligned} \bullet \text{ E step: } q_{i,y} &= \mu_y \exp \left[-\frac{(x_i - \theta^{(y)})^2}{2\sigma^2} \right] / \sum_{4=1}^k \mu_4 \exp \left[-\frac{(x_i - \theta^{(4)})^2}{2\sigma^2} \right] \\ \bullet \text{ M step: } \mu_4 &= \sum_{i=1}^n q_{i,4} / n \text{ and } \theta^{(4)} = \sum_{i=1}^n q_{i,4} x_i / \sum_{i=1}^n q_{i,4} \end{aligned}$$

Initialize $\theta^{(4)}$ and let $\mu_4 = 1/k$ ($4 = 1, \dots, k$)

iterate

// the E-step

```

for  $i = 1, \dots, n$ 
     $q_{i,y} = \mu_y p(x_i | \theta^{(y)}) / \sum_{k=1}^K \mu_k p(x_i | \theta^{(k)}) \quad (y = 1, \dots, k)$ 
end for
// the M-step
for  $y = 1, \dots, k$ 
     $\hat{\theta}^{(y)} = \arg \max_{\hat{\theta}} \sum_{i=1}^n q_{i,y} \ln p(x_i | \hat{\theta}) + \ln p(\hat{\theta})$ 
     $\mu_y = \sum_{i=1}^n q_{i,y} / n$ 
end for
until convergence

```

FIGURE 9.6 EM algorithm.

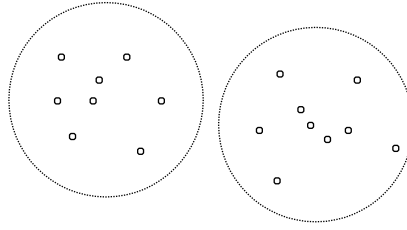


FIGURE 9.7 Gaussian mixture model with two mixture components.

9.6 Sequence Prediction Models

NLP problems involve sentences that can be regarded as sequences. For example, a sentence of n words can be represented as a sequence of n observations $\{x_1, \dots, x_n\}$. We are often interested in predicting a sequence of hidden labels $\{y_1, \dots, y_n\}$, one for each word. For example, in POS tagging, y_i is the POS of the word x_i .

The problem of predicting hidden labels $\{y_i\}$ given observations $\{x_i\}$ is often referred to as sequence prediction. Although this task may be regarded as a supervised learning problem, it has an extra complexity that data (x_i, y_i) in the sequence are dependent. For example, label y_i may depend on the previous label y_{i-1} . In the probabilistic modeling approach, one may construct a probability model of the whole sequence $\{(x_i, y_i)\}$, and then estimate the model parameters.

Similar to the standard supervised learning setting with independent observations, we have two types of models for sequence prediction: generative and discriminative. We shall describe both approaches in this section. For simplicity, we only consider first-order dependency where y_i only depends on y_{i-1} . Higher order dependency (e.g., y_i may depend on y_{i-2}, y_{i-3} , and so on) can be easily incorporated but requires more complicated notations. Also for simplicity, we shall ignore sentence boundaries, and just assume that the training data contain n sequential observations. In the following, we will assume that each y_i takes one of the k values in $\{1, \dots, k\}$.

9.6.1 Hidden Markov Model

The standard generative model for sequence prediction is the HMM, illustrated in Figure 9.8. It has been used in various NLP problems, such as POS tagging (Kupiec 1992). This model assumes that each y_i depends on the previous label y_{i-1} , and x_i only depends on y_i . Since x_i depends only on y_i , if the labels are observed on the training data, we may write the likelihood mathematically as

$$p(x_i | y_i, \theta) = p(x_i | \theta^{(n)}),$$

which is identical to (9.7).

One often uses the naive Bayes model for $p(x | \theta^{(\psi)})$. Because the observations x_i are independent conditioned on y_i , the parameter θ can be estimated from the training data using exactly the same method

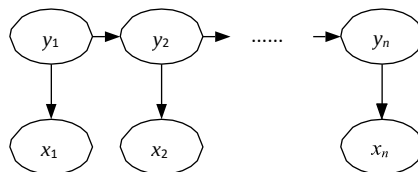


FIGURE 9.8 Graphical representation of HMM.

described in Section 9.4.1. Using the Bayes rule, the conditional probability of the label sequence $\{y_i\}$ is given by

$$p(\{y_i\}|\{x_i\}, \theta) \propto p(\{y_i\}) \prod_{i=1}^n p(x_i|\theta^{(y_i)}).$$

That is,

$$p(\{y_i\}|\{x_i\}, \theta) \propto \prod_{i=1}^n [p(x_i|\theta^{(y_i)})p(y_i|y_{i-1})]. \quad (9.16)$$

Similar to Section 9.4.1, the probability $p(y_i = a|y_{i-1} = b) = p(y_i = a, y_{i-1} = b)/p(y_{i-1} = b)$ can be estimated using counting. Let n_b be the number of training data with label b , and $n_{a,b}$ be the number of consecutive label pairs (y_i, y_{i-1}) with value (a, b) . We can estimate the conditional probability as

$$\begin{aligned} p(y_{i-1} = b) &= \frac{n_b}{n}, \\ p(y_i = a, y_{i-1} = b) &= \frac{n_{a,b}}{n}, \\ p(y_i = a|y_{i-1} = b) &= \frac{n_{a,b}}{n_b}. \end{aligned}$$

The process of estimating the sequence $\{y_i\}$ from observation $\{x_i\}$ is often called decoding. A standard method is the maximum likelihood decoding, which finds the most likely sequence $\{\hat{y}_i\}$ based on the conditional probability model (9.16). That is,

$$\{\{\hat{y}_i\}\} = \arg \max_{\{y_i\}} \prod_{i=1}^n f(y_i, y_{i-1}), \quad (9.17)$$

where

$$f(y_i, y_{i-1}) = \ln p(x_i|\theta^{(y_i)}) + \ln p(y_i|y_{i-1}).$$

It is not possible to enumerate all possible sequences $\{y_i\}$ and pick the largest score in (9.17) because the number of possible label sequences is k^n . However, an efficient procedure called the Viterbi decoding algorithm can be used to solve (9.17). The algorithm uses dynamic programming to track the best score up to a position j , and update the score recursively for $j = 1, \dots, n$. Let

$$s_j(y_j) = \max_{\{y\}} \prod_{i=1, \dots, j-1}^j f(y_i, y_{i-1}),$$

then it is easy to check that we have the following recursive identity:

$$s_{j+1}(y_{j+1}) = \max_{y_j \in \{1, \dots, k\}} [s_j(y_j) + f(y_{j+1}, y_j)].$$

Therefore, $s_j(y_j)$ can be computed recursively for $j = 1, \dots, n$. After computing $s_j(y_j)$, we may trace back $j = n, n-1, \dots, 1$ to find the optimal sequence $\{\hat{y}_j\}$. The Viterbi algorithm that solves (9.17) is presented in Figure 9.9.

```

Initialize  $s_0(y_0) = 0$  ( $y_0 = 1, \dots, k$ )
for  $j = 0, \dots, n - 1$ 
     $s_{j+1}(y_{j+1}) = \max_{y_j \in \{1, \dots, k\}} [s_j(y_j) + f(y_{j+1}, y_j)]$  ( $y_{j+1} = 1, \dots, k$ )
end for
 $\hat{y}_n = \arg \max_{y_n \in \{1, \dots, k\}} s_n(y_n)$ 
for  $j = n - 1, \dots, 1$ 
     $\hat{y}_j = \arg \max_{y_j \in \{1, \dots, k\}} [s_j(y_j) + f(\hat{y}_{j+1}, y_j)]$ 
end for

```

FIGURE 9.9 Viterbi algorithm.

9.6.2 Local Discriminative Model for Sequence Prediction

HMM is a generative model for sequence prediction. Similar to the standard supervised learning, one can also construct discriminative models for sequence prediction. In a discriminative model, in addition to the Markov dependency of y_i on y_{i-1} , we also allow an arbitrary dependency of y_i on $x^n = \{x_i\}_{i=1, \dots, n}$. That is, we consider a model of the form

$$p(\{y_i\} | x^n, \theta) = \prod_{i=1}^n p(y_i | y_{i-1}, x^n, \theta). \quad (9.18)$$

The graphical model representation is given in Figure 9.10.

One may use logistic regression (MaxEnt) to model the conditional probability in (9.18). That is, we let $\theta = w$ and

$$p(y_i | y_{i-1}, x^n, \theta) = \frac{\exp(w^T z_i(y_i, y_{i-1}, x^n))}{\sum_{k=1}^k \exp(w^T z_i(y_k, y_{i-1}, x^n))}. \quad (9.19)$$

The vector $z_i(y_i, y_{i-1}, x^n)$ is a human-constructed vector called feature vector. This model has identical form as the maximum entropy model (9.12). Therefore, supervised training algorithm for logistic regression can be directly applied to train the model parameter θ . On the test data, given a sequence x^n , one can use the Viterbi algorithm to decode $\{y_i\}$ using the scoring function $f_i(y_i, y_{i-1}) = \ln p(y_i | y_{i-1}, x^n, \theta)$. This method has been widely used in NLP, for example, POS tagging (Ratnaparkhi 1996).

More generally, one may reduce sequence prediction into a standard prediction problem, where we simply predict the next label y_i given the previous label y_{i-1} and the observation x_i^n . One may use any classification algorithm such as SVM to solve this problem. The scoring function returned by the

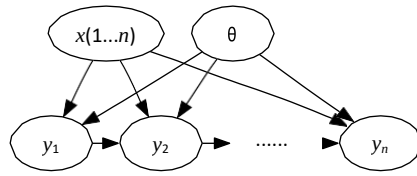


FIGURE 9.10 Graphical representation of discriminative local sequence prediction model.

underlying classifier can then be used as the scoring function for the Viterbi decoding algorithm. An example of this approach is given in Zhang et al. (2002).

9.6.3 Global Discriminative Model for Sequence Prediction

In (9.18), we decompose the conditional model of the label sequence $\{y_i\}$ using local model of the form $p(y_i|y_{i-1}, x^n, \theta)$ at each position i . Another approach is to treat the label sequence $y^n = \{y_i\}$ directly as a

multi-class classification problem with k^n possible values. We can then directly apply the MaxEnt model (9.12) to this k^n -class multi-category classification problem using the following representation:

$$p(y^n|w, x^n) = \frac{e^{f(w, x^n, y^n)}}{\sum_{y^n} e^{f(w, x^n, y^n)}}, \quad (9.20)$$

where

$$f(w, x^n, y^n) = \sum_{i=1}^n w^T z_i(y_i, y_{i-1}, x^n),$$

where $z_i(y_i, y_{i-1}, x^n)$ is a feature vector just like (9.19). While in (9.19), we model the local conditional probability $p(y_i|y_{i-1})$ that is a small fragment of the total label sequence $\{y_i\}$; in (9.20), we directly model the global label sequence.

The probability model (9.20) is called a conditional random field (Lafferty et al. 2001). The graphical model representation is given in Figure 9.11. Unlike Figure 9.10, the dependency between each y_i and y_{i-1} in Figure 9.11 is unidirectional. This means that we do not directly model the conditional dependency $p(y_i|y_{i-1})$, and do not normalize the conditional probability at each point i in the maximum entropy representation of the label sequence probability.

The CRF model is more difficult to train because the normalization factor in the denominator of (9.20) has to be computed in the training phase. Although the summation is over k^n possible values of the label sequence y^n , similar to the Viterbi decoding algorithm, the computation can be arranged efficiently using dynamic programming. In decoding, the denominator can be ignored in the maximum likelihood solution. That is, the most likely sequence $\{\hat{y}_i\}$ is the solution of

$$\{\hat{y}_i\} = \arg \max_{y^n} \sum_{i=1}^n w^T z_i(y_i, y_{i-1}, x^n). \quad (9.21)$$

The solution of this problem can be efficiently computed using the Viterbi algorithm.

More generally, global discriminative learning refers to the idea of treating sequence prediction as a multi-category classification problem with k^n classes, and a classification rule of the form (9.21). This approach can be used with some other learning algorithms such as Perceptron (Collins 2002) and large margin classifiers (Taskar et al. 2004; Tsochantaridis et al. 2005; Tillmann and Zhang 2008).

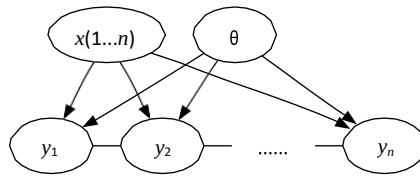


FIGURE 9.11 Graphical representation of a discriminative global sequence prediction model.

References

- Berger, A., S. A. Della Pietra, and V. J. Della Pietra, A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.
- Collins, M., Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Modeling (EMNLP'02)*, Philadelphia, PA, pp. 1–8, July 2002.
- Cortes, C. and V. N. Vapnik, Support vector networks. *Machine Learning*, 20:273–297, 1995.
- Dempster, A., N. Laird, and D. Rubin, Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.
- Hoerl, A. E. and R. W. Kennard, Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.
- Joachims, T., Text categorization with support vector machines: Learning with many relevant features. In *European Conference on Machine Learning, ECML-98*, Berlin, Germany, pp. 137–142, 1998.
- Kupiec, J., Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language*, 6:225–242, 1992.
- Lafferty, J., A. McCallum, and F. Pereira, Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML-01*, San Francisco, CA, pp. 282–289, 2001. Morgan Kaufmann.
- McCallum, A. and K. Nigam, A comparison of event models for naive Bayes text classification. In *AAAI/ICML-98 Workshop on Learning for Text Categorization*, Madison, WI, pp. 41–48, 1998.
- Ratnaparkhi, A., A maximum entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Philadelphia, PA, pp. 133–142, 1996.
- Taskar, B., C. Guestrin, and D. Koller, Max-margin Markov networks. In S. Thrun, L. Saul, and B. Schölkopf (editors), *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.
- Tillmann, C. and T. Zhang, An online relevant set algorithm for statistical machine translation. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(7):1274–1286, 2008.
- Tsochantaridis, I., T. Joachims, T. Hofmann, and Y. Altun, Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.
- Zhang, T., F. Damerau, and D. E. Johnson, Text chunking based on a generalization of Winnow. *Journal of Machine Learning Research*, 2:615–637, 2002.
- Zhang, T. and F. J. Oles, Text categorization based on regularized linear classification methods. *Information Retrieval*, 4:5–31, 2001.

10

Part-of-Speech Tagging

10.1	Introduction	205
	Parts of Speech · Part-of-Speech Problem	
10.2	The General Framework.....	208
10.3	Part-of-Speech Tagging Approaches	209
	Rule-Based Approaches · Markov Model Approaches · Maximum Entropy Approaches	
10.4	Other Statistical and Machine Learning Approaches.....	222
	Methods and Relevant Work · Combining Taggers	
10.5	POS Tagging in Languages Other Than English	225
	Chinese · Korean · Other Languages	
10.6	Conclusion	228
	References	229

Tunga Güngör
Boğaziçi University

10.1 Introduction

Computer processing of natural language normally follows a sequence of steps, beginning with a phoneme- and morpheme-based analysis and stepping toward semantics and discourse analyses. Although some of the steps can be interwoven depending on the requirements of an application (e.g., doing word segmentation and part-of-speech tagging together in languages like Chinese), dividing the analysis into distinct stages adds to the modularity of the process and helps in identifying the problems peculiar to each stage more clearly. Each step aims at solving the problems at that level of processing and feeding the next level with an accurate stream of data.

One of the earliest steps within this sequence is *part-of-speech* (POS) *tagging*. It is normally a sentence-based approach and given a sentence formed of a sequence of words, POS tagging tries to label (tag) each word with its correct part of speech (also named *word category*, *word class*, or *lexical category*). This process can be regarded as a simplified form (or a subprocess) of morphological analysis. Whereas morphological analysis involves finding the internal structure of a word (root form, affixes, etc.), POS tagging only deals with assigning a POS tag to the given surface form word. This is more true for Indo-European languages, which are the mostly studied languages in the literature. Other languages such as those from Uralic or Turkic families may necessitate a more sophisticated analysis for POS tagging due to their complex morphological structures.

10.1.1 Parts of Speech

A natural question that may arise is: what are these parts of speech, or how do we specify a set of

suitable parts of speech? It may be worthwhile at this point to say a few words about the origin of lexical categorization. From a linguistic point of view, the linguists mostly agree that there are three major (primary) parts of speech: noun, verb, and adjective (Pustet, 2003). Although there is some debate on the

205

topic (e.g., the claim that the adjective–verb distinction is almost nonexistent in some languages such as the East-Asian language Mandarin or the claim that all the words in a particular category do not show the same functional/semantic behavior), this minimal set of three categories is considered universal. The usual solution to the arguable nature of this set is admitting the inconsistencies within each group and saying that in each group there are “typical members” as well as not-so-typical members (Baker, 2003). For example, *eat* is a prototypical instance of the verb category because it describes a “process” (a widely accepted definition for verbs), whereas *hunger* is a less typical instance of a verb. This judgment is supported by the fact that *hunger* is also related to the adjective category because of the more common adjective *hungry*, but there is no such correspondence for *eat*.

Taking the major parts of speech (noun, verb, adjective) as the basis of lexical categorization, linguistic models propose some additional categories of secondary importance (adposition, determiner, etc.) and some subcategories of primary and secondary categories (Anderson, 1997; Taylor, 2003). The subcategories either involve distinctions that are reflected in the morphosyntax (such as tense or number) or serve to capture different syntactic and semantic behavioral patterns (such as for nouns, count-noun and mass-noun). In this way, while the words in one subcategory may undergo some modifications, the others may not.

Leaving aside these linguistic considerations and their theoretical implications, people in the realm of natural language processing (NLP) approach the issue from a more practical point of view. Although the decision about the size and the contents of the *tagset* (the set of POS tags) is still linguistically oriented, the idea is providing distinct parts of speech for all classes of words having distinct grammatical behavior, rather than arriving at a classification that is in support of a particular linguistic theory. Usually the size of the tagset is large and there is a rich repertoire of tags with high discriminative power. The most frequently used corpora (for English) in the POS tagging research and the corresponding tagsets are as follows: Brown corpus (87 basic tags and special indicator tags), Lancaster-Oslo/Bergen (LOB) corpus (135 tags of which 23 are base tags), Penn Treebank and Wall Street Journal (WSJ) corpus (48 tags of which 12 are for punctuation and other symbols), and Susanne corpus (353 tags).

10.1.2 Part-of-Speech Problem

Except a few studies, nearly all of the POS tagging systems presuppose a fixed tagset. Then, the problem is, given a sentence, assigning a POS tag from the tagset to each word of the sentence. There are basically two difficulties in POS tagging:

1. *Ambiguous words.* In a sentence, obviously there exist some words for which more than one POS tag is possible. In fact, this language property makes POS tagging a real problem, otherwise the solution would be trivial. Consider the following sentence:

We can can the can

The three occurrences of the word *can* correspond to auxiliary, verb, and noun categories, respectively. When we take the whole sentence into account instead of the individual words, it is easy to determine the correct role of each word. It is easy at least for humans, but may not be so for automatic taggers. While disambiguating a particular word, humans exploit several mechanisms and information sources such as the roles of other words in the sentence, the syntactic structure of the sentence, the domain of the text, and the commonsense knowledge. The problem for computers is finding out how to handle all this information.

2. *Unknown words.* In the case of rule-based approaches to the POS tagging problem that use a set of handcrafted rules, there will clearly be some words in the input text that cannot be handled by the rules. Likewise, in statistical systems, there will be words that do not appear in the training corpus. We call such words *unknown words*. It is not desirable from a practical point of view for a tagger to adopt a closed-world assumption — considering only the words and sentences from which the

rules or statistics are derived and ignoring the rest. For instance, a syntactic parser that relies on

the output of a POS tagger will encounter difficulties if the tagger cannot say anything about the unknown words. Thus, having some special mechanisms for dealing with unknown words is an important issue in the design of a tagger.

Another issue in POS tagging, which is not directly related to language properties but poses a problem for taggers, is the consistency of the tagset. Using a large tagset enables us to encode more knowledge about the morphological and morphosyntactical structures of the words, but at the same time makes it more difficult to distinguish between similar tags. Tag distinctions in some cases are so subtle that even humans may not agree on the tags of some words. For instance, an annotation experiment performed in Marcus et al. (1993) on the Penn Treebank has shown that the annotators disagree on 7.2% of the cases on the average. Building a consistent tagset is a more delicate subject for morphologically rich languages since the distinctions between different affix combinations need to be handled carefully. Thus, we can consider the inconsistencies in the tagsets as a problem that degrades the performance of taggers.

A number of studies allow some ambiguity in the output of the tagger by labeling some of the words with a set of tags (usually 2–3 tags) instead of a single tag. The reason is that, since POS tagging is seen as a preprocessing step for other higher-level processes such as named-entity recognition or syntactic parsing, it may be wiser to output a few most probable tags for some words for which we are not sure about the correct tag (e.g., both of the tags *IN** and *RB* may have similar chances of being selected for a particular word). This decision may be left to later processing, which is more likely to decide on the correct tag by exploiting more relevant information (which is not available to the POS tagger).

The state-of-the-art in POS tagging accuracy (number of correctly tagged word tokens over all word tokens) is about 96%–97% for most Indo-European languages (English, French, etc.). Similar accuracies are obtained for other types of languages provided that the characteristics different from Indo-European languages are carefully handled by the taggers. We should note here that it is possible to obtain high accuracies using very simple methods. For example, on the WSJ corpus, tagging each word in the test data with the most likely tag for that word in the training data gives rise to accuracies around 90% (Halteren et al., 2001; Manning and Schütze, 2002). So, the sophisticated methods used in the POS tagging domain and that will be described throughout the chapter are for getting the last 10% of tagging accuracy.

On the one hand, 96%–97% accuracy may be regarded as quite a high success rate, when compared with other NLP tasks. Based on this figure, some researchers argue that we can consider POS tagging as an already-solved problem (at least for Indo-European languages). Any performance improvement above these success rates will be very small. However, on the other hand, the performances obtained with current taggers may seem insufficient and even a small improvement has the potential of significantly increasing the quality of later processing. If we suppose that a sentence in a typical English text has 20–30 words on the average, an accuracy rate of 96%–97% implies that there will be about one word erroneously tagged per sentence. Even one such word will make the job of a syntax analyzer much more difficult. For instance, a rule-based bottom-up parser begins from POS tags as the basic constituents and at each step combines a sequence of constituents into a higher-order constituent. A word with an incorrect tag will give rise to an incorrect higher-order structure and this error will probably affect the other constituents as the parser moves up in the hierarchy. Independent of the methodology used, any syntax analyzer will exhibit a similar behavior. Therefore, we may expect a continuing research effort on POS tagging.

This chapter will introduce the reader to a wide variety of methods used in POS tagging and with the solutions of the problems specific to the task. Section 10.2 defines the POS tagging problem and describes the approach common to all methods. Section 10.3 discusses in detail the main formalisms used in the

* In most of the examples in this chapter, we will refer to the tagset of the Penn Treebank. The tags that appear in the chapter

are CD (cardinal number), DT (determiner), IN (preposition or subordinating conjunction), JJ (adjective), MD (modal verb), NN (noun, singular or mass), NNP (proper noun, singular), NNS (noun, plural), RB (adverb), TO (to), VB (verb, base form), VBD (verb,

past tense), *VBG* (verb, gerund or present participle), *VCN* (verb, past participle), *VBP* (verb, present, non-3rd personsingular), *VBZ* (verb, present, 3rd personsingular), *WDT* (wh-determiner), and *WP* (wh-pronoun).

domain. Section 10.4 is devoted to a number of methods used less frequently by the taggers. Section 10.5 discusses the POS tagging problem for languages other than English. Section 10.6 concludes this chapter.

10.2 The General Framework

Let $W = w_1 w_2 \dots w_n$ be a sentence having n words. The task of POS tagging is finding the set of tags $T = t_1 t_2 \dots t_n$, where t_i corresponds to the POS tag of w_i , $1 \leq i \leq n$, as accurately as possible. In determining the correct tag sequence, we make use of the morphological and syntactic (and maybe semantic) relationships within the sentence (the *context*). The question is how a tagger encodes and uses the constraints enforced by these relationships.

The traditional answer to this question is simply limiting the context to a few words around the *target word* (the word we are trying to disambiguate), making use of the information supplied by these words and their tags, and ignoring the rest. So, if the target word is w_i , a typical context comprises w_{i-2} , t_{i-2} , w_{i-1} , t_{i-1} , and w_i . (Most studies scan the sentence from left to right and use the information on the already-tagged left context. However, there are also several studies that use both left and right contexts.) The reason for restricting the context severely is being able to cope with the exponential nature of the problem. As we will see later, adding one more word into the context increases the size of the problem (e.g., the number of parameters estimated in a statistical model) significantly.

It is obvious that long-distance dependencies between the words also play a role in determining the POS tag of a word. For instance, in the phrase

the girls can...

the tag of the underlined word can is ambiguous: it may be an auxiliary (e.g., the girls can do it) or a verb (e.g., the girls can the food). However, if we use a larger context instead of only one or two previous words, the tag can be uniquely determined:

The man who saw the girls can...

In spite of several such examples, it is customary to use a limited context in POS tagging and similar problems. As already mentioned, we can still get quite high success rates.

In the case of unknown words, the situation is somewhat different. One approach is again resorting to the information provided by the context words. Another approach that is more frequently used in the literature is making use of the morphology of the target word. The morphological data supplied by the word typically include the prefixes and the suffixes (more generally, a number of initial and final characters) of the word, whether the word is capitalized or not, and whether it includes a hyphen or not. For example, as an initial guess, Brill (1995a) assigns the tag proper noun to an unknown word if it is capitalized and the tag common noun otherwise. As another example, the suffix *-ing* for an unknown word is a strong indication for placing it in the verb category.

There are some studies in the POS tagging literature that are solely devoted to the tagging of unknown words (Mikheev, 1997; Thede, 1998; Nagata, 1999; Cucerzan and Yarowsky, 2000; Lee et al., 2002; Nakagawa and Matsumoto, 2006). Although each one uses a somewhat different technique than the others, all of them exploit the contextual and morphological information as already stated. We will not

directly cover these studies explicitly in this chapter; instead, we will mention the issues relevant to unknown word handling within the explanations of tagging algorithms.

10.3 Part-of-Speech Tagging Approaches

10.3.1 Rule-Based Approaches

The earliest POS tagging systems are rule-based systems, in which a set of rules is manually constructed and then applied to a given text. Probably the first rule-based tagging system is given by Klein and Simpson (1963), which is based on a large set of handcrafted rules and a small lexicon to handle the exceptions. The initial tagging of the Brown corpus was also performed using a rule-based system, TAGGIT (Manning and Schütze, 2002). The lexicon of the system was used to constrain the possible tags of a word to those that exist in the lexicon. The rules were then used to tag the words for which the left and right context words were unambiguous. The main drawbacks of these early systems are the laborious work of manually coding the rules and the requirement of linguistic background.

10.3.1.1 Transformation-Based Learning

A pioneering work in rule-based tagging is by Brill (1995a). Instead of trying to acquire the linguistic rules manually, Brill (1995a) describes a system that learns a set of correction rules by a methodology called *transformation-based learning* (TBL). The idea is as follows: First, an initial-state annotator assigns a tag to each word in the corpus. This initial tagging may be a simple one such as choosing one of the possible tags for a word randomly, assigning the tag that is seen most often with a word in the training set, or just assuming each word as a noun (which is the most common tag). It can also be a sophisticated scheme such as using the output of another tagger. Following the initialization, the learning phase begins. By using a set of predetermined *rule templates*, the system instantiates each template with data from the corpus (thus obtaining a set of *rules*), applies temporarily each rule to the incorrectly tagged words in the corpus, and identifies the best rule that reduces most the number of errors in the corpus. This rule is added to the set of learned rules. Then the process iterates on the new corpus (formed by applying the selected rule) until none of the remaining rules reduces the error rate by more than a prespecified threshold.

The rule templates refer to a context of words and tags in a window of size seven (the target word, three words on the left, and three words on the right). Each template consists of two parts, a triggering environment (if-part) and a rewrite rule (action):

Change the tag (of the target word) from A to B if condition

It becomes applicable when the condition is satisfied. An example template referring to the previous tag and an example instantiation of it (i.e., a rule) for the sentence `the can rusted` are given below (`can` is the target word whose current tag is `modal`):

Change the tag from A to B if the previous tag is X

Change the tag from `modal` to `noun` if the previous tag is `determiner`

The rule states that the current tag of the target word that follows a `determiner` is `modal` but the correct tag must be `noun`. When the rule is applied to the sentence, it actually corrects one of the errors and increases its chance of being selected as the best rule.

Table 10.1 shows the rule templates used in Brill (1995a) and Figure 10.1 gives the TBL algorithm. In the algorithm, C_k refers to the training corpus at iteration k and M is the number of words in the corpus. For a rule r , r_e , r_{t_1} , and r_{t_2} correspond to the triggering environment, the left tag in the rule action, and the right tag in the rule action, respectively (i.e., “change the tag from r_{t_1} to r_{t_2} if r_e ”). For a word w_i , $w_{i,e}$, $w_{i,c}$, and $w_{i,t}$ denote the environment, the current tag, and the correct tag, respectively. The function $f(e)$ is a binary function that returns 1 when the expression e evaluates to true and 0 otherwise. $C_k(r)$ is the result of applying rule r to the corpus at iteration k . R is the set of learned rules. The first statement inside the loop calculates, for each rule r , the number of times it corrects an incorrect tag and the number of times it changes a correct tag to an incorrect one, whenever its triggering environment matches the environment

of the target word. Subtracting the first quantity from the second gives the amount of error reduction by

TABLE 10.1 Rule Templates Used in Transformation-Based Learning

Change the tag from A to B if		
$t_{i-1} = X$	$t_{i-2} = X \text{ and } t_{i+1} = Y \text{ } w_i = X \text{ and } t_{i+1} = Y$	
$t_{i+1} = X$	$w_{i-1} = X$	$w_i = X$
$t_{i-2} = X$	$w_{i+1} = X$	$w_{i-1} = X \text{ and } t_{i-1} = Y$
$t_{i+2} = X$	$w_{i-2} = X$	$w_{i-1} = X \text{ and } t_{i+1} = Y$
$t_{i-2} = X \text{ or } t_{i-1} = X$	$w_{i+2} = X$	$t_{i-1} = X \text{ and } w_{i+1} = Y$
$t_{i+1} = X \text{ or } t_{i+2} = X$	$w_{i-2} = X \text{ or } w_{i-1} = X$	$w_{i+1} = X \text{ and } t_{i+1} = Y$
$t_{i-3} = X \text{ or } t_{i-2} = X \text{ or } t_{i-1} = X \text{ } w_{i+1} = X \text{ or } w_{i+2} = X \text{ } w_{i-1} = X \text{ and } t_{i-1} = Y \text{ and } w_i = Z \text{ } t_{i+1} = X \text{ or } t_{i+2} = X \text{ or } t_{i+3} = X \text{ } w_{i-1} = X \text{ and } w_i = Y \text{ } w_{i-1} = X \text{ and } w_i = Y \text{ and } t_{i+1} = Z \text{ } t_{i-1} = X \text{ and } t_{i+1} = Y$		
$= X \text{ and } t_{i+2} = Y \text{ } t_{i-1} = X \text{ and } w_i = Y \text{ } w_i = X \text{ and } w_{i+1} = Y \text{ and } t_{i-1} = X \text{ and } w_i = Y \text{ and } w_{i+1} = Z \text{ } t_{i-1} = X \text{ and } w_i = Y \text{ } w_i = X \text{ and } w_{i+1} = Y \text{ and } t_{i+1} = Z$		

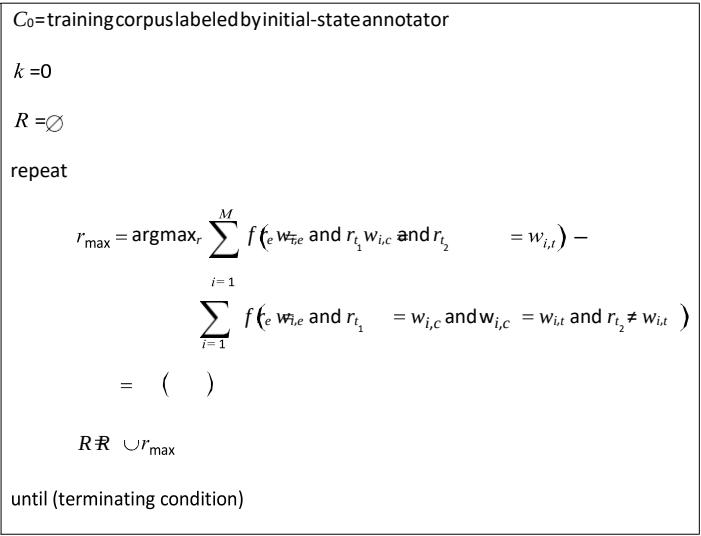


FIGURE 10.1 Transformation-based learning algorithm.

this rule and we select the rule with the largest error reduction. Some example rules that were learned by the system are the following:

- Change the tag from VB to NN if one of the previous two tags is DT
- Change the tag from NN to VB if the previous tag is TO
- Change the tag from VBP to VB if one of the previous two words is n' t

The unknown words are handled in a similar manner, with the following two differences: First, since no information exists for such words in the training corpus, the initial-state annotator assigns the tag proper noun if the word is capitalized and the tag common noun otherwise. Second, the templates use morphological information about the word, rather than contextual information. Two templates used by the system are given below together with example instantiations:

Change the tag from A to B if the last character of the word is X

Change the tag from A to B if character X appears in the word

Change the tag from NN to NNS if the last character of the word is -s (e.g., tables)

Change the tag from NN to CD if character. appears in the word (e.g., 10.42)

The TBL tagger was trained and tested on the WSJ corpus, which uses the Penn Treebank tagset. The system learned 447 contextual rules (for known words) and 243 rules for unknown words. The accuracy

was 96.6% (97.2% for known words and 82.2% for unknown words). There are a number of advantages of TBL over some of the stochastic approaches:

- Unlike hidden Markov models, the system is quite flexible in the features that can be incorporated into the model. The rule templates can make use of any property of the words in the environment.
- Stochastic methods such as hidden Markov models and decision lists can overfit the data. However, TBL seems to be more immune from such overfitting, probably because of learning on the whole dataset at each iteration and its logic behind ordering the rules (Ramshaw and Marcus, 1994; Carberry et al., 2001).
- The output of TBL is a list of rules, which are usually easy to interpret (e.g., a determiner is most likely followed by a noun rather than a verb), instead of a huge number of probabilities as in other models.

It is also possible to use TBL in an unsupervised manner, as shown in Brill (1995b). In this case, by using a dictionary, the initial-state annotator assigns all possible tags to each word in the corpus. So, unlike the previous approach, each word will have a set of tags instead of a single tag. Then, the rules try to reduce the ambiguity by eliminating some of the tags of the ambiguous words. We no longer have rule templates that replace a tag with another tag; instead, the templates serve to reduce the set of tags to a singleton:

Change the tag from A to B if condition

where A is a set of tags and $B \in A$. We determine the most likely tag B by considering each element of A in turn, looking at each context in which this element is unambiguous, and choosing the most frequently occurring element. For example, given the following sentence and knowing that the word can (underlined) is either MD, NN, or VB,

The/DT can/MD, NN, VB is/VBZ open/JJ

we can infer the tag NN for can if the unambiguous words in the context DT _ VBZ are mostly NN. Note that the system takes advantage of the fact that many words have only one tag and thus uses the unambiguous contexts when scoring the rules at each iteration.

Some example rules learned when the system was applied to the WSJ corpus are given below:

- Change the tag from {NN, VB, VBP} to VBP if the previous tag is NNS
- Change the tag from {NN, VB} to VB if the previous tag is MD
- Change the tag from {JJ, NNP} to JJ if the following tag is NNS

The system was tested on several corpora and it achieved accuracies up to 96.0%, which is quite a high accuracy for an unsupervised method.

10.3.1.2 Modifications to TBL and Other Rule-Based Approaches

The transformation-based learning paradigm and its success in the POS tagging problem have influenced many researchers. Following the original publication, several extensions and improvements have been proposed. One of them, named *guaranteed pre-tagging*, analyzes the effect of fixing the initial tag of those words that we already know to be correct (Mohammed and Pedersen, 2003). Unlike the standard TBL tagger, if we can identify the correct tag of a word a priori and give this information to the tagger, then the tagger initializes the word with this “pre-tag” and guarantees that it will not be changed during learning. However, this pre-tag can still be used in any contextual rule for changing the tags of other words. The rest of the process is the same as in the original algorithm. Consider the word chair (underlined) in the following sentence, with the initial tags given as shown:

Mona/NNP will/MD sit/VB in/IN the/DT pretty/RB chair/NN this/DT time/NN

The standard TBL tagger will change the tag of chair to VB due to a learned rule: “change the tag from

NN to VB if the following tag is DT.” Not only the word `chair` will be incorrectly tagged, also the initial

incorrect tag of the word `pretty` will remain unchanged. However, if we have a priori information that `chair` is being used as `NN` in this particular context, then it can be pre-tagged and will not be affected by the mentioned rule. Moreover, the tag of `pretty` will be corrected due to the rule “change the tag from `RB` to `JJ` if the following tag is `NN`.”

The authors developed the guaranteed pre-tagging approach during a word sense disambiguation task on Senseval-2 data. There were about 4300 words in the dataset that were manually tagged. When the standard TBL algorithm was executed to tag all the words in the dataset, the tags of about 570 of the manually tagged words (which were the correct tags) were changed. This motivated the pre-tagged version of the TBL algorithm. The manually tagged words were marked as pre-tagged and the algorithm did not allow these tags to be changed. This caused 18 more words in the context of the pre-tagged words to be correctly tagged.

The main drawback of the TBL approach is its high time complexity. During each pass through the training corpus, it forms and evaluates all possible instantiations of every suitable rule template. (We assume the original TBL algorithm as we have described here. The available version in fact contains some optimizations.) Thus, when we have a large corpus and a large set of templates, it becomes intractable. One solution to this problem is putting a limit on the number of rules (instantiations of rule templates) that are considered for incorrect taggings. The system developed by Carberry et al. (2001), named *randomized TBL*, is based on this idea: at each iteration, it examines each incorrectly tagged word, but only R (a predefined constant) of all possible template instantiations that would correct the tag are considered (randomly selected). In this way, the training time becomes independent of the number of rules.

Even with a very low value for R (e.g., $R = 1$), the randomized TBL obtains, in much less time, an accuracy very close to that of the standard TBL. This may seem interesting, but has a simple explanation. During an iteration, the standard TBL selects the best rule. This means that this rule corrects many incorrect tags in the corpus. So, although randomized TBL considers only R randomly generated rules on each instance, the probability of generating this particular rule will be high since it is applicable to many incorrect instances. Therefore, these two algorithms tend to learn the same rules at early phases of the training. In later phases, since the rules will be less applicable to the remaining instances (i.e., more specific rules), the chance of learning the same rules decreases. Even if randomized TBL cannot determine the best rule at an iteration, it can still learn a compensating rule at a later iteration.

The experiments showed the same success rates for both versions of TBL, but the training time of randomized TBL was 5–10 times better. As the corpus size decreases, the accuracy of randomized TBL becomes slightly worse than the standard TBL, but the time gain becomes more impressive.

Finite state representations have a number of desirable properties, like efficiency (using a deterministic and minimized machine) and the compactness of the representation. In Roche and Schabes (1995), it was attempted to convert the TBL POS tagging system into a finite state transducer (FST). The idea is that, after the TBL algorithm learns the rules in the training phase, the test (tagging) phase can be done much more efficiently. Given a set of rules, the FST tagger is constructed in four steps: converting each rule (contextual rule or unknown word rule) into an FST; globalizing each FST so that it can be applied to the whole input in one pass; composing all transducers into a single transducer; and determinizing the transducer.

The method takes advantage of the well-defined operations on finite state transducers — composing, determinizing, and minimizing. The lexicon, which is used by the initial-state annotator, is also converted into a finite state automaton. The experiments on the Brown corpus showed that the FST tagger runs much faster than both the TBL tagger (with the same accuracy) and their implementation of a trigram-based stochastic tagger (with a similar accuracy).

Multidimensional transformation-based learning (mTBL) is a framework where TBL is applied to more than one task jointly. Instead of learning the rules for different tasks separately, it may be beneficial to acquire them in a single learning phase. The motivation under the mTBL framework is exploiting the dependencies between the tasks and thus increasing the performance on the individual tasks. This idea

was applied to POS tagging and text chunking (identification of basic phrasal structures) (Florian and

Ngai, 2001). The mTBL algorithm is similar to the TBL algorithm, except that the objective function used to select the best rule is changed as follows:

$$f(r) = - \sum_{s \in \text{corpus}} \sum_{i=1}^n w_i * (S_i(r(s)) - S_i(s))$$

where

r is a rule

n is the number of tasks (2, in this application)

$r(s)$ denotes the application of rule r to sample s in the corpus

$S_i(\cdot)$ is the score of task i (1: correct, 0: incorrect)

w_i is a weight assigned to task i (used to weight the tasks according to their importance).

The experiments on the WSJ corpus showed about 0.5% increase in accuracy

Below we show the rules learned in the jointly trained system and in the POS-tagging-only system for changing VBD tag to VBN. The former one learns a single rule (a more general rule), indicating that if the target word is inside a verb phrase then the tag should be VBN. However, the latter system can arrive at this decision using three rules. Since the rules are scored separately in the standard TBL tagger during learning, a more general rule in mTBL will have a better chance to capture the similar incorrect instances.

Change the tag from VBD to VBN if the target chunk is I-VP

Change the tag from VBD to VBN if one of the previous three tags is VBZ

Change the tag from VBD to VBN if the previous tag is VBD

Change the tag from VBD to VBN if one of the previous three tags is VBP

While developing a rule-based system, an important issue is determining in which order the rules should be applied. There may be several rules applicable for a particular situation and the output of the tagger may depend on in which order the rules are applied. A solution to this problem is assigning some weights (votes) to the rules according to the training data and disambiguating the text based on these votes (Tür and Oflazer, 1998). Each rule is in the following form:

$$(c_1, c_2, \dots, c_n; v)$$

where

$c_i, 1 \leq i \leq n$, is a constraint that incorporates POS and/or lexical (word form) information of the words in the context

v is the vote of the rule

Two example rule instantiations are:

([tag=MD], [tag=RB], [tag=VB]; 100)

([tag=DT, lex=that], [tag=NNS]; -100)

The first one promotes (a high positive vote) a modal followed by a verb with an intervening adverb and the second one demotes a singular determiner reading of *that* before a plural noun. The votes are acquired automatically from the training corpus by counting the frequencies of the patterns denoted by the constraints. As the votes are obtained, the rules are applied to the possible tag sequences of a sentence and the tag sequence that results in the maximum vote is selected. The method of applying rules to an input sentence resembles the Viterbi algorithm commonly used in stochastic taggers. The proposed

method, therefore, can also be approached from a probabilistic point of view as selecting the best tag sequence among all possible taggings of a sentence.

A simple but interesting technique that is different from context-based systems is learning the rules from word endings (Grzymala-Busse and Old, 1997). This is a word-based approach (not using information from context) that considers a fixed number of characters (e.g., three) at the end of the words. A table is built from the training data that lists all word endings that appear in the corpus, accompanied with the correct POS. For instance, the sample list of four entries

(-ine, noun) (-inc, noun) (-ing, noun) (-ing, verb)

implies noun category for -ine and -inc, but signals a conflict for -ing. The table is fed to a rule induction algorithm that learns a set of rules by taking into account the conflicting cases. The algorithm outputs a particular tag for each word ending. A preliminary experiment was done by using Roget's dictionary as the training data. The success is low as might be expected from such an information-poor approach: about 26% of the words were classified incorrectly.

10.3.2 Markov Model Approaches

The rule-based methods used for the POS tagging problem began to be replaced by stochastic models in the early 1990s. The major drawback of the oldest rule-based systems was the need to manually compile the rules, a process that requires linguistic background. Moreover, these systems are not robust in the sense that they must be partially or completely redesigned when a change in the domain or in the language occurs. Later on a new paradigm, statistical natural language processing, has emerged and offered solutions to these problems. As the field became more mature, researchers began to abandon the classical strategies and developed new statistical models.

Several people today argue that statistical POS tagging is superior to rule-based POS tagging. The main factor that enables us to use statistical methods is the availability of a rich repertoire of data sources: lexicons (may include frequency data and other statistical data), large corpora (preferably annotated), bilingual parallel corpora, and so on. By using such resources, we can learn the usage patterns of the tag sequences and make use of this information to tag new sentences. We devote the rest of this section and the next section to statistical POS tagging models.

10.3.2.1 The Model

Markov models (MMs) are probably the most studied formalisms in the POS tagging domain. Let $W = w_1 w_2 \dots w_n$ be a sequence of words and $T = t_1 t_2 \dots t_n$ be the corresponding POS tags. The problem is finding the optimal tag sequence corresponding to the given word sequence and can be expressed as maximizing the following conditional probability:

$$P(T|W)$$

Applying Bayes' rule, we can write

$$P(T|W) = \frac{P(W|T)P(T)}{P(W)}$$

The problem of finding the optimal tag sequence can then be stated as follows:

$$\arg \max_T P(T|W) = \arg \max_T \frac{P(W|T)P(T)}{P(W)}$$

$$= \arg \max_T P(W|T)P(T) \quad (10.1)$$

where the $P(W)$ term was eliminated since it is the same for all T . It is impracticable to directly estimate the probabilities in Equation 10.1, therefore we need some simplifying assumptions. The first term $P(W|T)$

can be simplified by assuming that the words are independent of each other given the tag sequence (Equation 10.2) and a word only depends on its own tag (Equation 10.3):

$$P(W|T) = P(w_1 \dots w_n | t_1 \dots t_n)$$

$$= \prod_{i=1}^n P(w_i | t_1 \dots t_n) \quad (10.2)$$

$$= \prod_{i=1}^n P(w_i | t_i) \quad (10.3)$$

The second term $P(T)$ can be simplified by using the limited horizon assumption, which states that a tag depends only on k previous tags (k is usually 1 or 2):

$$P(T) = P(t_1 \dots t_n)$$

$$= P(t_1)P(t_2 | t_1)P(t_3 | t_1 t_2) \dots P(t_n | t_1 \dots t_{n-1})$$

$$= \prod_{i=1}^n P(t_i | t_{i-1} \dots t_{i-k})$$

When $k = 1$, we have a *first-order model (bigram model)* and when $k = 2$, we have a *second-order model (trigram model)*. We can name $P(W|T)$ as the *lexical probability* term (it is related to the lexical forms of the words) and $P(T)$ as the *transition probability* term (it is related to the transitions between tags). Now we can restate the POS tagging problem: finding the tag sequence $T = t_1 t_2 \dots t_n$ (among all

possible tag sequences) that maximizes the lexical and transition probabilities:

$$\operatorname{argmax}_T \prod_{i=1}^n P(w_i | t_i) P(t_i | t_{i-1} \dots t_{i-k}) \quad (10.4)$$

This is a hidden Markov model (HMM) since the tags (states of the model) are hidden and we can only observe the words. Having a corpus annotated with POS tags (*supervised tagging*), the training phase (estimation of the probabilities in Equation 10.4) is simple using maximum likelihood estimation:

$$P(w_i | t_i) = \frac{f(w_i, t_i)}{f(t_i)} \quad \text{and} \quad P(t_i | t_{i-1} \dots t_{i-k}) = \frac{f(t_{i-k} \dots t_i)}{f(t_{i-k} \dots t_{i-1})}$$

where

$f(w, t)$ is the number of occurrences of word w with tag t

$f(t_1 \dots t_m)$ is the number of occurrences of the tag sequence $t_1 \dots t_m$

That is, we compute the relative frequencies of tag sequences and word-tag pairs from the training data.

Then, in the test phase (tagging phase), given a sequence of words W , we need to determine the tag sequence that maximizes these probabilities as shown in Equation 10.4. The simplest approach may be computing Equation 10.4 for each possible tag sequence of length n and then taking the maximizing sequence. Clearly, this naive approach yields an algorithm that is exponential in the number of words. This problem can be solved more efficiently using dynamic programming techniques and a well-known dynamic programming algorithm used in POS tagging and similar tasks is the Viterbi algorithm (see Chapter 9). The Viterbi algorithm, instead of keeping track of all paths during execution, determines the optimal subpaths for each node while it traverses the network and discards the others. It is an efficient algorithm operating in linear time.

The process described above requires an annotated corpus. Though such corpora are available for well-studied languages, it is difficult to find such resources for most of the other languages. Even when an annotated corpus is available, a change of the domain (i.e., training on available annotated corpus and testing on a text from a new domain) causes a significant decrease in accuracy (e.g., Boggess et al., 1999). However, it is also possible to learn the parameters of the model without using an annotated training dataset (*unsupervised tagging*). A commonly used technique for this purpose is the expectation maximization method.

Given training data, the forward-backward algorithm, also known as the Baum-Welch algorithm, adjusts the parameter probabilities of the HMM to make the training sequence as likely as possible (Manning and Schütze, 2002). The forward-backward algorithm is a special case of the expectation maximization method. The algorithm begins with some initial probabilities for the parameters (transitions and word emissions) we are trying to estimate and calculates the probability of the training data using these probabilities. Then the algorithm iterates. At each iteration, the probabilities of the parameters that are on the paths that are traversed more by the training data are increased and the probabilities of other parameters are decreased. The probability of the training data is recalculated using this revised set of parameter probabilities. It can be shown that the probability of the training data increases at each step. The process repeats until the parameter probabilities converge. Provided that the training dataset is representative of the language, we can expect the learned model to behave well on the test data. After the parameters are estimated, the tagging phase is exactly the same as in the case of supervised tagging.

In general, it is not possible to observe all the parameters in Equation 10.4 in the training corpus for all words w_i in the language and all tags t_i in the tagset, regardless of how large the corpus is. During testing, when an unobserved term appears in a sentence, the corresponding probability and thus the probability of the whole sentence will be zero for a particular tag sequence. This is named *sparse data problem* and is a problem for all probabilistic methods. To alleviate this problem, some form of smoothing is applied. A smoothing method commonly used in POS taggers is linear interpolation, as shown below for a second-order model:

$$P(t_i|t_{i-1}t_{i-2}) \cong \lambda_1 P(t_i) + \lambda_2 P(t_i|t_{i-1}) + \lambda_3 P(t_i|t_{i-1}t_{i-2})$$

where the λ_i 's are constants with $0 \leq \lambda_i \leq 1$ and $\sum \lambda_i = 1$. That is, unigram and bigram data are also considered in addition to the trigrams. Normally, λ_i 's are estimated from a development corpus, which is distinct from the training and the test corpora. Some other popular smoothing methods are discounting and back-off, and their variations (Manning and Schütze, 2002).

10.3.2.2 HMM-Based Taggers

Although it is not entirely clear who first used MMs for the POS tagging problem, the earliest account in the literature appears to be Bahl and Mercer (1976). Another early work that popularized the idea of statistical tagging is due to Church (1988), which uses a standard MM and a simple smoothing technique. Following these works, a large number of studies based on MMs were proposed. Some of these use the standard model (the model depicted in Section 10.3.2.1) and play with a few properties (model order, smoothing, etc.) to improve the performance. Some others, on the other hand, in order to overcome the limitations posed by the standard model, try to enrich the model by making use of the context in a different manner, modifying the training algorithm, and so on.

A comprehensive analysis of the effect of using MMs for POS tagging was given in an early work by Merialdo (1994). In this work, a second-order model is used in both a supervised and an unsupervised manner. An interesting point of this study is the comparison of two different schemes in finding the optimal tag sequence of a given (test) sentence. The first one is the classical Viterbi approach as we

have explained before, called “sentence level tagging” in Merialdo (1994). An alternative is “word level

tagging" which, instead of maximizing over the possible tag sequences for the sentence, maximizes over the possible tags for each word:

$$\arg \max_T P(T|W) \text{ vs. } \arg \max_{t_i} P(t_i|W)$$

This distinction was considered in Dermatas and Kokkinakis (1995) as well and none of these works observed a significant difference in accuracy under the two schemes. To the best of our knowledge, this issue was not analyzed further and later works relied on Viterbi tagging (or its variants).

Merialdo (1994) uses a form of interpolation where trigram distributions are interpolated with uniform distributions. A work that concentrates on smoothing techniques in detail is given in Sündermann and Ney (2003). It employs linear interpolation and proposes a new method for learning λ_i 's that is based on the concept of training data coverage (number of distinct n -grams in the training set). It argues that using a large model order (e.g., five) accompanied with a good smoothing technique has a positive effect on the accuracy of the tagger. Another example of a sophisticated smoothing technique is given in Wang and Schuurmans (2005). The idea is exploiting the similarity between the words and putting similar words into the same cluster. Similarity is defined in terms of the left and right contexts. Then, the parameter probabilities are estimated by averaging, for a word w , over probabilities of 50 most similar words of w .

It was shown empirically in Dermatas and Kokkinakis (1995) that the distribution of the unknown words is similar to that of the less probable words (words occurring less than a threshold t , e.g., $t = 10$). Therefore, the parameters for the unknown words can be estimated from the distributions of less probable words. Several models were tested, particularly first- and second-order HMMs were compared with a simpler model, named *Markovian language model* (MLM), in which the lexical probabilities $P(W|T)$ are ignored. All the experiments were repeated on seven European languages. The study arrives at the conclusion that HMM reduces the error almost to half in comparison to the same order MLM.

A highly accurate and frequently cited (partly due to its availability) POS tagger is the TnT tagger (Brants, 2000). Though based on the standard HMM formalism, its power comes from a careful treatment of smoothing and unknown word issues. The smoothing is done by context-independent linear interpolation. The distribution of unknown words is estimated using the sequences of characters at word endings, with sequence length varying from 1 to 10. Instead of considering all the words in the training data while determining the similarity of an unknown word to other words, only the infrequent ones (occurring less than 10 times) are taken into account. This is in line with the justification in Dermatas and Kokkinakis (1995) about the similarity between unknown words and less probable words. Another interesting property is the incorporation of capitalization feature in the tagset. It was observed that the probability distributions of tags around capitalized words are different from those around lowercased words. So, each tag is accompanied with a capitalization feature (e.g., instead of VBD, VBDC and VBDC'), doubling the size of the tagset. To increase the efficiency of the tagger, beam search is used in conjunction with the Viterbi algorithm, which prunes the paths more while scanning the sentence. The TnT tagger achieves about 97% accuracy on the Penn Treebank.

Some studies attempted to change the form of Equation 10.4 in order to incorporate more context into the model. Thede and Harper (1999) change the lexical probability $P(w_i|t_i)$ to $P(w_i|t_{i-1}, t_i)$ and also use a similar formula for unknown words, $P(w_i \text{ has suffix } s|t_{i-1}, t_i)$, where the suffix length varies from 1 to 4. In a similar manner, Banko and Moore (2004) prefer the form $P(w_i|t_{i-1}, t_i, t_{i+1})$. The authors of these two works name their modified models as *full second-order HMM* and *contextualized HMM*, respectively. In Lee et al. (2000a), more context is considered by utilizing the following formulations:

$$P(T) \cong \prod_{i=1}^n P(t_i|t_{i-1} \dots t_{i-k}, w_{i-1} \dots w_{i-l})$$

$$i=1$$

$$n$$

$$-$$

$$P(W|T)$$

$$\cong$$

$$P(w_i | t_{i-1} \dots t_{i-l}, t_i, w_{i-1} \dots w_{i-l}) \tag{10.5}$$

$i=1$

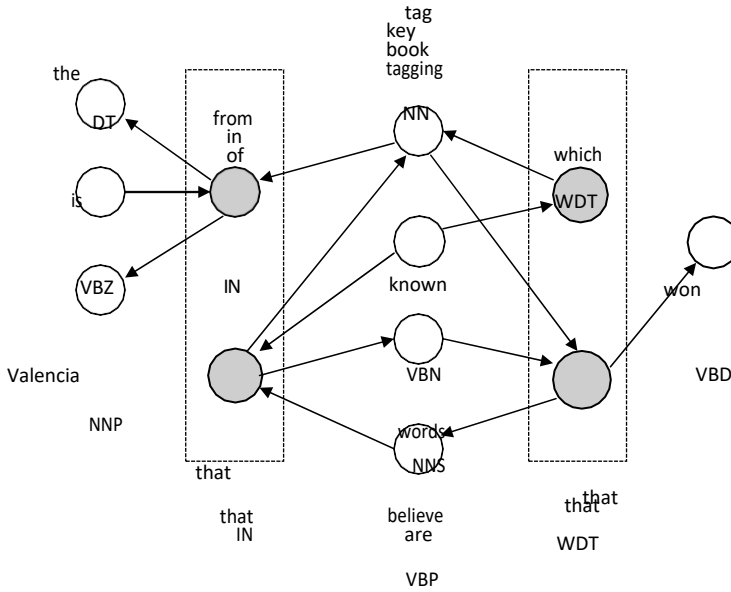


FIGURE 10.2 A part of an example HMM for the specialized word *that*. (Reprinted From Pla, F. and Molina, A., *Nat. Lang. Eng.*, 10, 167, 2004. Cambridge University Press. With permission.)

The proposed model was investigated using several different values for the parameters K , J , L , and I (between 0 and 2). In addition, the conditional distributions in Equations 10.5 were converted into joint distributions. This formulation was observed to yield more reliable estimations in such extended contexts. The experimental results obtained in all these systems showed an improvement in accuracy compared to the standard HMM.

A more sophisticated way of enriching the context is identifying a priori a set of “specialized words” and, for each such word w , splitting each state t in the HMM that emits w into two states: one state (w, t) that only emits w and another state, the original state t , that emits all the words emitted before splitting it except w (Pla and Molina, 2004). In this way, the model can distinguish among different local contexts. An example for a first-order model is given in Figure 10.2, where the dashed rectangles show the split states.

The specialized words can be selected using different strategies: words with high frequencies in the training set, words that belong to closed-class categories, or words resulting in a large number of tagging errors on a development set. The system developed uses the TnT tagger. The evaluation using different numbers of specialized words showed that the method gives better results than HMM (for all numbers of specialized words, ranging from 1 to 400), and the optimum performance was obtained with about 30 and 285 specialized words for second- and first-order models, respectively.

In addition to the classical view of considering each word and each tag in the dataset separately, there exist some studies that combine individual words/tags in some manner. In Cutting et al. (1992), each word is represented by an ambiguity class, which is the set of its possible parts of speech. In Nasr et al. (2004), the tagset is extended by adding new tags, the so-called ambiguous tags. When a word in a certain context can be tagged as $t_1, t_2, \dots, t_k$ with probabilities that are close enough, an ambiguous tag $t_{1,2,\dots,k}$ is

created. In such cases, instead of assigning the tag with the highest score to the word in question, it seems desirable to allow some ambiguity in the output, since the tagger is not sure enough about the correct tag. For instance, the first five ambiguous tags obtained from the Brown corpus are IN-RB, DT-IN-WDT-WP, JJ-VBN, NN-VB, and JJ-NN. Success rates of about 98% were obtained with an ambiguity of 1.23 tags/word.

Variable memory Markov models (VMMM) and self-organizing Markov models (SOMM) were proposed as solutions to the POS tagging problem (Schütze and Singer, 1994; Kim et al., 2003). They aim at increasing the flexibility of the HMMs by being able to vary the size of the context as the need arises (Manning and Schütze, 2002). For instance, the VMMM can go from a state that considers the previous

two tags to a state that does not use any context, then to a state that uses the previous three tags. This differs from linear interpolation smoothing which always uses a weighted average of a fixed number of n -grams. In both VMMM and SOMM, the structure of the model is induced from the training corpus. Kim et al. (2003) represent the MM in terms of a statistical decision tree (SDT) and give an algorithm for learning the SDT. These extended MMs yield results comparable to those of HMMs, with significant reductions in the number of parameters to be estimated.

10.3.3 Maximum Entropy Approaches

The HMM framework has two important limitations for classification tasks such as POS tagging: strong independence assumptions and poor use of contextual information. For HMM POS tagging, we usually assume that the tag of a word does not depend on previous and next words, or a word in the context does not supply any information about the tag of the target word. Furthermore, the context is usually limited to the previous one or two words. Although there exist some attempts to overcome these limitations, as we have seen in Section 10.3.2.2, they do not allow us to use the context in any way we like.

Maximum entropy (ME) models provide us more flexibility in dealing with the context and are used as an alternative to HMMs in the domain of POS tagging. The use of the context is in fact similar to that in the TBL framework. A set of *feature templates* (in analogy to rule templates in TBL) is predefined and the system learns the discriminating features by instantiating the feature templates using the training corpus. The flexibility comes from the ability to include any template that we think useful—may be simple (target tag t_i depends on t_{i-1}) or complex (t_i depends on t_{i-1} and/or t_{i-2} and/or w_{i+1}). The features need not be independent of each other and the model exploits this advantage by using overlapping and interdependent features.

A pioneering work in ME POS tagging is Ratnaparkhi (1996, 1998). The probability model is defined over $H \times T$, where H is the set of possible contexts (histories) and T is the set of tags. Then, given $h \in H$ and $t \in T$, we can express the conditional probability in terms of a log-linear (exponential) model:

$$P(t|h) = \frac{1}{Z(h)} \prod_{j=1}^k f_j^{f_j(t,h)}$$

where

$$Z(h) = \sum_{t \in T} \prod_{j=1}^k \alpha_j^{f_j(t,h)}$$

$f_1, \dots, f_k$ are the features, $\alpha_j > 0$ is the “weight” of feature f_j , and $Z(h)$ is a normalization function to ensure a true probability distribution. Each feature is binary-valued, that is, $f_j(t, h) = 0$ or 1 . Thus, the probability $P(t|h)$ can be interpreted as the normalized product of the weights of the “active” features on (t, h) .

The probability distribution P we seek is the one that maximizes the entropy of the distribution under some constraints:

$$\begin{aligned} & \arg \max_P - \sum_{h \in H} \sum_{t \in T} P(t|h) \log P(t|h) \\ & \text{subject to} \end{aligned}$$

$$\bar{P}(h)P(t|h)\log P(t|h)$$

$$E(f_j) = \tilde{E}(f_j), \quad 1 \leq j \leq k$$

TABLE 10.2 Feature Templates Used in the Maximum Entropy Tagger

Condition	Features	
For all words w_i	$t_{i-1} = X$	and $t_i = T$
	$t_{i-2} = X$ and $t_{i-1} = Y$	and $t_i = T$
	$w_{i-1} = X$	and $t_i = T$
	$w_{i-2} = X$	and $t_i = T$
	$w_{i+1} = X$	and $t_i = T$
	$w_{i+2} = X$	and $t_i = T$
Word w_i is not a rare word	$w_i = X$	and $t_i = T$
Word w_i is a rare word	X is a prefix of w_i , $ X \leq 4$	and $t_i = T$
	X is a suffix of w_i , $ X \leq 4$	and $t_i = T$
	w_i contains number	and $t_i = T$
	w_i contains uppercase character	and $t_i = T$
	w_i contains hyphen	and $t_i = T$

Source: Ratnaparkhi, A., A maximum entropy model for part-of-speech tagging, in *EMNLP*, Brill, E. and Church, K. (eds.), ACL, Philadelphia, PA, 1996, 133–142. With permission.

where

$$E(f_i) = \frac{1}{n} \sum_{i=1}^n \bar{P}(h_i) P(t_i|h_i) f_i(h_i, t_i)$$
$$\bar{E}(f_i) = \frac{1}{n} \sum_{i=1}^n \bar{P}(h_i, t_i) f_i(h_i, t_i)$$

$E(f_i)$ and $\bar{E}(f_i)$ denote, respectively, the model’s expectation and the observed expectation of feature f_i . $\bar{P}(h_i)$ and $\bar{P}(h_i, t_i)$ are the relative frequencies, respectively, of context h_i and the context-tag pair (h_i, t_i) in the training data. The intuition behind maximizing the entropy is that it gives us the most uncertain distribution. In other words, we do not include any information in the distribution that is not justified by the empirical evidence available to us. The parameters of the distribution P can be obtained using the generalized iterative scaling algorithm (Darroch and Ratcliff, 1972).

The feature templates used in Ratnaparkhi (1996) are shown in Table 10.2. As can be seen, the context (history) is formed of $(w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}, t_{i-2}, t_{i-1})$, although it is possible to include other data. The features for rare words (words occurring less than five times) make use of morphological clues such as affixes and capitalization. During training, for each target word w , the algorithm instantiates each feature template by using the context of w . For example, two features that can be extracted from the training corpus are shown below:

$$f_{\mathcal{F}}(h_i, t_i) = \begin{cases} 1 & \text{if } t_{i-1} = \text{JJ and } t_i = \text{NN} \\ 0 & \text{else} \end{cases}$$

$$f_{\mathcal{F}}(h_i, t_i) = \begin{cases} 1 & \text{if suffix}(w_{i-1}) = \text{--ing and } t_i = \text{VBG} \\ 0 & \text{else} \end{cases}$$

The features that occur rarely in the data are usually unreliable and they do not have much predictive power. The algorithm uses a simple smoothing technique and eliminates those features that appear less than a threshold (e.g., less than 10 times). There are some other studies that use more sophisticated smoothing methods, such as using a Gaussian prior on the model parameters, which improve the performance when compared with the frequency cutoff technique (Curran and Clark, 2003; Zhao et al., 2007).

In the test phase, beam search is used to find the most likely tag sequence of a sentence. If a dictionary is available, each known word is restricted to its possible tags in order to increase the efficiency. Otherwise, all tags in the tagset become candidates for the word.

Experiments on the WSJ corpus showed 96.43% accuracy. In order to observe the effect of the flexible feature selection capability of the model, the author analyzed the problematic words (words frequently mistagged) and added more specialized features into the model to better distinguish such cases. A feature about the problematic word *about* may be:

$$f(h_i, t) = \begin{cases} 1 & \text{if } w_i = \text{"about"} \text{ and } t_{i-2}t_{i-1} = \text{DT NNS and } t_i = \text{IN} \\ 0 & \text{else} \end{cases}$$

An insignificant increase was observed in the accuracy (96.49%). This 96%–97% accuracy barrier may be partly due to missing some information that is important for disambiguation or to the inconsistencies in the training corpus. In the work, this argument was also tested by repeating the experiments on more consistent portions of the corpus and the accuracy increased to 96.63%. We can thus conclude that, as mentioned by several researchers, there is some amount of noise in the corpora and this seems to prevent taggers passing beyond an accuracy limit.

It is worth noting before closing this section that there are some attempts for detecting and correcting the inconsistencies in the corpora (Květon and Oliva, 2002a,b; Dickinson and Meurers, 2003; Rocío et al., 2007). The main idea in these attempts is determining (either manually or automatically) the tag sequences that are impossible or very unlikely to occur in the language, and then replacing these sequences with the correct ones after a manual inspection. For example, in English, it is nearly impossible for a determiner to be followed by a verb. As the errors in the annotated corpora are reduced via such techniques, we can expect the POS taggers to obtain better accuracies.

10.3.3.1 Taggers Based on ME Models

The flexibility of the feature set in the ME model has been exploited in several ways by researchers. Toutanova and Manning (2000) concentrate on the problematic cases for both unknown/rare words and known words. Two new feature templates are added to handle the unknown and rare words:

- A feature activated when all the letters of a word are uppercase
- A feature activated when a word that is not at the beginning of the sentence contains an uppercase letter

In fact, these features reflect the peculiarities in the particular corpus used, the WSJ corpus. In this corpus, for instance, the distribution of words in which only the initial letter is capitalized is different from the distribution of words whose all letters are capitalized. Thus, such features need not be useful in other corpora. Similarly, in the case of known words, the most common error types are handled by using new feature templates. An example template is given below:

VBD/VBN ambiguity — a feature activated when there is *have* or *be* auxiliary form in the preceding eight positions

All these features are corpus- and language-dependent, and may not generalize easily to other situations. However, these specialized features show us the flexibility of the ME model.

Some other works that are built upon the models of Ratnaparkhi (1996) and Toutanova and Manning (2000) use bidirectional dependency networks (Toutanova et al., 2003; Tsuruoka and Tsujii, 2005). Unlike previous works, the information about the future tags is also taken into account and both left and right contexts are used simultaneously. The justification can be given by the following example:

will to fight...

When tagging the word *will*, the tagger will prefer the (incorrect but most common) modal sense if only the left context (which is empty in this example) is examined. However, if the word on the right (*to*) is also included in the context, the fact that *to* is often preceded by a noun will force the correct tag for the word *will*. A detailed analysis of several combinations of left and right contexts reveal some useful results: the left context always carries more information than the right context, using both contexts increases the success rates, and symmetric use of the context is better than using (the same amount of) only left or right context (e.g., $t_{i-1}t_{i+1}$ is more informative than $t_{i-2}t_{i-1}$ and $t_{i+1}t_{i+2}$). Another strategy analyzed in Tsuruoka and Tsujii (2005) is called the easiest-first strategy, which, instead of tagging a sentence in left-to-right order, begins from the “easiest word” to tag and selects the easiest word among the remaining words at each step. The easiest word is defined as the word whose probability estimate is the highest. This strategy makes sense since a highly ambiguous word forced to be tagged early and tagged incorrectly will degrade the performance and it may be wiser to leave such words to the final steps where more information is available.

All methodologies used in POS tagging make the stationary assumption that the position of the target word within the sentence is irrelevant to the tagging process. However, this assumption is not always realistic. For example, when the word *walk* appears at the front of a sentence it usually indicates a physical exercise (corresponding to noun tag) and when it appears toward the end of a sentence it denotes an action (verb tag), as in the sentences:

A morning walk is a blessing for the whole day
It only takes me 20 minutes to walk to work

By relaxing this stationary assumption, a formalism called *nonstationary maximum entropy Markov model* (NS-MEMM), which is a generalization of the MEMM framework (McCallum et al., 2000), was proposed in Xiao et al. (2007). The model is decomposed into two component models, the n -gram model and the ME model:

$$P(t|h) = P(t|t^j)P_{ME}(t|h) \quad (10.6)$$

where t^j denotes a number of previous tags (so, $P(t|t^j)$ corresponds to the transition probability). In order to incorporate position information into the model, sentences are divided into k bins such that the i th word of a sentence of length n takes part in (approximately) the $\lfloor \frac{i}{k} \rfloor$ th bin. For instance, for a 20-word sentence and $k = 4$, the first five words will be in the first bin, and so on. This additional parameter introduced into the model obviously increases the dimensionality of the model. Equation 10.6 is thus modified to include the position parameter p :

$$P(t|h, p) = P(t|t^j, p)P_{ME}(t|h, p)$$

The experiments on three corpora showed improvement over the ME model and the MEMM. The number of bins ranged from 1 (ordinary MEMM) to 8. A significant error reduction was obtained for $k = 2$ and $k = 3$; beyond this point the behavior was less predictable.

Curran et al. (2006) employ ME tagging in a multi-tagging environment. As in other studies that preserve some ambiguity in the final tags, a word is assigned all the tags whose probabilities are within a factor of the probability of the most probable tag. To account for this, the forward-backward algorithm is adapted to the ME framework. During the test phase, a word is considered to be tagged correctly if the correct tag appears in the set of tags assigned to the word. The results on the CCGbank corpus show an accuracy of 99.7% with 1.40 tags/word.

10.4 Other Statistical and Machine Learning Approaches

There are a wide variety of learning paradigms in the machine learning literature (Alpaydın, 2004). However, the learning approaches other than the HMMs have not been used so widely for the POS

tagging problem. This is probably due to the suitability of the HMM formalism to this problem and the high success rates obtained with HMMs in early studies. Nevertheless, all well-known learning paradigms have been applied to POS tagging in some degree. In this section, we list these approaches and cite a few typical studies that show how the tagging problem can be adapted to the underlying framework. The interested reader should refer to this chapter's section in the companion wiki for further details.

10.4.1 Methods and Relevant Work

- *Support vector machines.* Support vector machines (SVM) have two advantages over other models: they can easily handle high-dimensional spaces (i.e., large number of features) and they are usually more resistant to overfitting (see Nakagawa et al., 2002; Mayfield et al., 2003).
- *Neural networks.* Although neural network (NN) taggers do not in general seem to outperform the HMM taggers, they have some attractive properties. First, ambiguous tagging can be handled easily without additional computation. When the output nodes of a network correspond to the tags in the tagset, normally, given an input word and its context during the tagging phase, the output node with the highest activation is selected as the tag of the word. However, if there are several output nodes with close enough activation values, all of them can be given as candidate tags. Second, neural network taggers converge to top performances with small amounts of training data and they are suitable for languages for which large corpora are not available (see Schmid, 1994; Roth and Zelenko, 1998; Marques and Lopes, 2001; Pérez-Ortiz and Forcada, 2001; Raju et al., 2002).
- *Decision trees.* Decision trees (DT) and statistical decision trees (SDT) used in classification tasks, similar to rule-based systems, can cover more context and enable flexible feature representations, and also yield outputs easier to interpret. The most important criterion for the success of the learning algorithms based on DTs is the construction of a set of questions to be used in the decision procedure (see Black et al., 1992; Màrquez et al., 2000).
- *Finite state transducers.* Finite state machines are efficient devices that can be used in NLP tasks that require a sequential processing of inputs. In the POS tagging domain, the linguistic rules or the transitions between the tag states can be expressed in terms of finite state transducers (see Roche and Schabes, 1995; Kempe, 1997; Grăna et al., 2003; Villamil et al., 2004).
- *Genetic algorithms.* Although genetic algorithms have accuracies worse than those of HMM taggers and rule-based approaches, they can be seen as an efficient alternative in POS tagging. They reach performances near their top performances with small populations and a few iterations (see Araujo, 2002; Alba et al., 2006).
- *Fuzzy set theory.* The taggers formed using the fuzzy set theory are similar to HMM taggers, except that the lexical and transition probabilities are replaced by fuzzy membership functions. One advantage of these taggers is their high performances with small data sizes (see Kim and Kim, 1996; Kogut, 2002).
- *Machine translation ideas.* An approach used recently in the POS tagging domain and that is on a different track was inspired by the ideas used in machine translation (MT). Some of the works consider the sentences to be tagged as belonging to the source language and the corresponding tag sequences as the target language, and apply statistical machine translation techniques to find the correct "translation" of each sentence. Other works aim at discovering a mapping from the taggings in a source language (or several source languages) to the taggings in a target language. This is a useful approach when there is a shortage of annotated corpora or POS taggers for the target language (see Yarowsky et al., 2001; Fossum and Abney, 2005; Finch and Sumita, 2007; Mora and Peiró, 2007).
- *Others.* Logical programming (see Cussens, 1998; Lager and Nivre, 2001; Reiser and Riddle, 2001), dynamic Bayesian networks and cyclic dependency networks (see Peshkin et al., 2003; Reynolds

and Bilmes, 2005; Tsuruoka et al., 2005), memory-based learning (see Daelemans et al., 1996),

relaxation labeling (see Padró, 1996), robust risk minimization (see Ando, 2004), conditional random fields (see Lafferty et al., 2001), Markov random fields (see Jung et al., 1996), and latent semantic mapping (see Bellegarda, 2008).

It is worth mentioning here that there has also been some work on *POS induction*, a task that aims at dividing the words in a corpus into different categories such that each category corresponds to a part of speech (Schütze, 1993; Schütze, 1995; Clark, 2003; Freitag, 2004; Rapp, 2005; Portnoy and Bock, 2007). These studies mainly use clustering algorithms and rely on the distributional characteristics of the words in the text. The task of POS tagging is based on a predetermined tagset and therefore adopts the assumptions it embodies. However, this may not be appropriate always, especially when we are using texts from different genres or from different languages. So, labeling the words with tags that reflect the characteristics of the text in question may be better than trying to label with an inappropriate set of tags. In addition, POS induction has a cognitive science motivation in the sense that it aims at showing how the evidence in the linguistic data can account for language acquisition.

10.4.2 Combining Taggers

As we have seen in the previous sections, the POS tagging problem was approached using different machine learning techniques and 96%–97% accuracy seems a performance barrier for almost all of them. A question that may arise at this point is whether we can obtain better results by combining different taggers and/or models. It was observed that, although different taggers have similar performances, they usually produce different errors (Brill and Wu, 1998; Halteren et al., 2001). Based on this encouraging observation, we can benefit from using more than one tagger in such a way that each individual tagger deals with the cases where it is the best.

One way of combining taggers is using the output of one of the systems as input to the next system. An early application of this idea is given in Tapanainen and Voutilainen (1994), where a rule-based system first reduces the ambiguities in the initial tags of the words as much as possible and then an HMM-based tagger arrives at the final decision. The intuition behind this idea is that rules can resolve only some of the ambiguities but with a very high correctness and the stochastic tagger resolves all ambiguities but with a lower accuracy. The method proposed in Clark et al. (2003) is somewhat different and it investigates the effect of co-training, where two taggers are iteratively retrained on each other's output. The taggers should be sufficiently different (e.g., based on different models) for co-training to be effective. This approach is suitable in cases when there is a small amount of annotated corpora. Beginning from a seed set (annotated sentences), both of the taggers (T1 and T2) are trained initially. Then the taggers are used to tag a set of unannotated sentences. The output of T1 is added to the seed set and used to retrain T2; likewise, the output of T2 is added to the seed set to retrain T1. The process is repeated using a new set of unannotated sentences at each iteration.

The second way in combining taggers is letting each tagger to tag the same data and selecting one of the outputs according to a voting strategy. Some of the common voting strategies are given in Brill and Wu (1998); Halteren et al. (2001); Mihalcea (2003); Glass and Bangay (2005); Yonghui et al. (2006):

- *Simple voting*. The tag decided by the largest number of the taggers is selected (by using an appropriate method for breaking the ties).
- *Weighted voting 1*. The decisions of the taggers are weighted based on their general performances, that is, the higher the accuracy of a tagger, the larger its weight.
- *Weighted voting 2*. This is similar to the previous one, except that the performance on the target word (certainty of the tagger on the current situation) is used as the weight instead of the general performance.
- *Ranked voting*. This is similar to the weighted voting schemes, except that the ranks (1, 2, etc.) of

the taggers are used as weights, where the best tagger is given the highest rank.

The number of taggers in the combined tagger normally ranges from two to five and they should have different structures for an effective combination. Except Glass and Bangay (2005), the mentioned studies observed an improvement in the success rates. Glass and Bangay (2005) report that the accuracies of the combined taggers are in between the accuracies of the best and the worst individual taggers, and it is not always true that increasing the number of taggers yields better results (e.g., a two-tagger combination may outperform a five-tagger combination). The discouraging results obtained in this study may be partly due to the peculiarities of the domain and the tagset used. Despite this observation, we can in general expect a performance increase by the combination of different taggers.

10.5 POS Tagging in Languages Other Than English

As in other fields of NLP, most of the research on POS tagging takes English as the language of choice. The motivation in this choice is being able to compare the proposed models (new models or variations of existing models) with previous work. The success of stochastic methods largely depends on the availability of language resources — lexicons and corpora. Beginning from 1960s, such resources have begun to be developed for the English language (e.g., Brown corpus). This availability enabled the researchers to concentrate on the modeling issue, rather than the data issue, in developing more sophisticated approaches.

However, this is not the case for other (especially non-Indo-European) languages. Until recently there was a scarcity of data sources for these languages. As new corpora begin to appear, research attempts in the NLP domain begin to increase. In addition, these languages have different morphological and syntactic characteristics than English. A naive application of a POS tagger developed with English in mind may not always work. Therefore, the peculiarities of these languages should be taken into account and the underlying framework should be adapted to these languages while developing POS taggers.

In this section, we first concentrate on two languages (that do not belong to the Indo-European family) that are widely studied in the POS tagging domain. The first one, Chinese, is typical in its word segmentation issues; the other one, Korean, is typical in its agglutinative nature. We briefly mention the characteristics of these languages from a tagging perspective. Then we explain the solutions to these issues proposed in the literature.

There are plenty of research efforts related to POS tagging of other languages. These studies range from sophisticated studies for well-known languages (e.g., Spanish, German) to those in primitive stages of development (e.g., for Vietnamese). The works in the first group follow a similar track as those for English. They exploit the main formalisms used in POS tagging and adapt these strategies to the particular languages. We have not included these works in previous parts of this chapter and instead we have mostly considered the works on English, because of being able to do a fair comparison between methodologies. The works in the second group are usually in the form of applying the well-known models to those languages.

10.5.1 Chinese

A property of the Chinese language that makes POS tagging more difficult than languages such as English is that the sentences are written without spaces between the characters. For example, two possible segmentations of the underlined part of the sentence

他俩儿谈恋爱是从头年元月开始的

are

(a)	...	是		从头		年		元月	
		V		ADV		TIME-CLASSIFIER		TIME-N	
(b)	...	是		从		头年		元月	...
		V		PREP		TIME-N		TIME-N	

Since POS tagging depends on how the sentence is divided into words, a successful word segmentation is a prerequisite for a tagger. In some works on Chinese POS tagging, a correctly segmented word sequence is assumed as input. However, this may not always be a realistic assumption and a better approach is integrating these two tasks in such a way that any one of them may contribute to the success of the other. For instance, a particular segmentation that seems as the best one to the word segmentation component may be rejected due to its improper tagging. Another property of the Chinese language is the difference of its morphological and syntactic structures. Chinese grammar focuses on the word order rather than the morphological variations. Thus, transition information contributes more to POS tagging than morphological information. This property also indicates that unknown word processing should be somewhat different from English-like languages.

The works in Sun et al. (2006) and Zhou and Su (2003) concentrate on integrating word segmentation and POS tagging. Given a sentence, possible segmentations and all possible taggings for each segmentation are taken into account. Then the most likely path, a sequence of (word, tag) pairs, is determined using a Viterbi-like algorithm. Accuracies about 93%–95% were obtained, where it was measured in terms of both correctly identified segments and tags. Zhang and Clark (2008) formulate the word segmentation and POS tagging tasks as a single problem, take the union of the features of each task as the features of the joint system, and apply the perceptron algorithm of Collins (2002). Since the search space formed of combined (word, tag) pairs is very large, a novel multiple beam search algorithm is used, which keeps track of a list of candidate parses for each character in the sentence and thus avoids limiting the search space as in previous studies. A comparison with the two-stage (word segmentation followed by POS tagging) system showed an improvement of about 10%–15% in F-measure.

Maximum entropy framework is used in Zhao et al. (2007) and Lin and Yuan (2002). Since the performance of the ME models is sensitive to the features used, some features that take the characteristics of the language into account are included in the models. An example of an HMM-based system is given in Cao et al. (2005). Instead of using the probability distributions in the standard HMM formalism, it combines the transition and lexical probabilities as

$$\arg \max_T \prod_{i=1}^n P(t_i, w_i | t_{i-1}, w_{i-1})$$

and then converts into the following form to alleviate data sparseness:

$$\arg \max_T \prod_{i=1}^n P(t_i | t_{i-1}, w_{i-1}) P(w_i | t_{i-1}, w_{i-1}, t_i)$$

A tagger that combines rule-based and HMM-based processes in a cascaded manner is proposed in

Ning et al. (2007). It first reduces the ambiguity in the initial assignment of the tags by employing a TBL-like process. Then HMM training is performed on this less ambiguous data. The accuracy results for Chinese POS tagging are around 92%–94% for open test (test data contains unknown words) and 96%–98% for closed test (no unknown words in the test data). Finally, we should mention an interesting study that is about Classical Chinese, which has some grammatical differences from Modern Chinese (Huang et al., 2002).

10.5.2 Korean

Korean, which belongs to the group of Altaic languages, is an agglutinative language and has a very productive morphology. In theory, the number of possible morphological variants of a given word can be in tens of thousands. For such languages, a word-based tagging approach does not work due to the sparse data problem. Since there exist several surface forms corresponding to a base form, the number of out-of-vocabulary words will be very large and the estimates from the corpus will not be reliable. A common solution to this problem is morpheme-based tagging: each morpheme (either a base form or an affix) is tagged separately. Thus, the problem of POS tagging changes into the problem of morphological tagging (morphological disambiguation) for agglutinative languages: we tag each morpheme separately and then combine. As an example, Figure 10.3 shows the morpheme structure of the Korean sentence *na-neun hag-gyo-e gan-da* (I go to school). Straight lines indicate the word boundaries, dashed lines indicate the morpheme boundaries, and the correct tagging is given by the thick lines.

The studies in Lee et al. (2000b), Lee and Rim (2004), and Kang et al. (2007) apply n -gram and HMM models to the Korean POS tagging problem. For instance, Lee et al. (2000b) propose the following morpheme-based version of the HMM model:

$$\prod_{i=1}^u P(c_i, p_i | c_{i-1} \dots c_{i-K}, p_{i-1} \dots p_{i-L}, m_{i-1} \dots m_{i-J}) P(m_i | c_i \dots c_{i-L}, p_i \dots p_{i-L+1}, m_{i-1} \dots m_{i-J}) \quad (10.7)$$

where

u is the number of morphemes

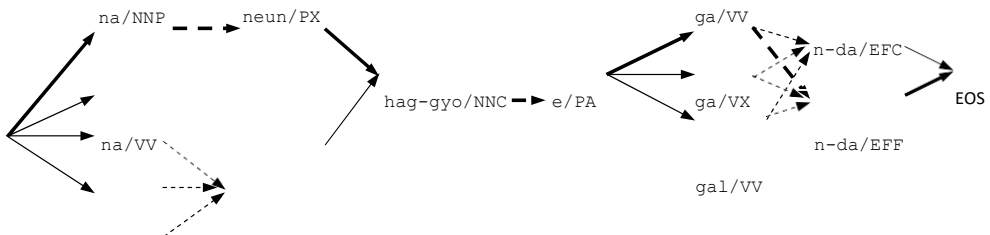
c denotes a (morpheme) tag

m is a morpheme

p is a binary parameter (e.g., 0 and 1) differentiating transitions across a word boundary and transitions within a word

The indices K, J, L and I range from 0 to 2. In fact, Equation 10.7 is analogous to a word-based HMM equation if we regard m as word (w) and c as tag (t) (and ignore p 's).

Han and Palmer (2005) and Ahn and Seo (2007) combine statistical methods with rule-based disambiguation. In Ahn and Seo (2007), different sets of rules are used to identify the idiomatic constructs and to resolve the ambiguities in highly ambiguous words. The rules eliminate some of the taggings and then an HMM executes in order to arrive at the final tag sequence. When a word is inflected in Korean, the base form of the word and/or the suffix may change their forms (by character deletion or by contraction), forming allomorphs. Before POS tagging, Han and Palmer (2005) attempt to recover the original forms of the words and the suffixes by using rule templates extracted from the corpus. Then an n -gram approach tags the given sentence in the standard way. The accuracies obtained by these works are between 94% and 97%.



BOS na/VX neun/EFD

na/VV

FIGURE 10.3 Morpheme structure of the sentence *na-neun hag-gyo-e gan-da*. (From Lee, D. and Rim, H., Part-of-speech tagging considering surface form for an agglutinative language, in *Proceedings of the ACL*, ACL, Barcelona, Spain, 2004. With permission.)

10.5.3 Other Languages

We can cite the following works related to POS tagging for different language families and groups. By no means we claim that the groups presented below are definite (this is a profession of linguists) nor the languages included are exhaustive. We simply mention some worth-noting studies in a wide coverage of languages. The interested readers can refer to the cited references.

- *Indo-European languages*. Spanish (Triviño-Rodríguez and Morales-Bueno, 2001; Jiménez and Morales, 2002; Carrasco and Gelbukh, 2003), Portuguese (Lopes and Jorge, 2000; Kepler and Finger, 2006), Dutch (Prins, 2004; Poel et al., 2007), Swedish (Eineborg and Lindberg, 2000), Greek (Maragoudakis et al., 2004).
- *Agglutinative and inflectional languages*. Japanese (Asahara and Matsumoto, 2000; Ma, 2002), Turkish (Altunyurt et al., 2007; Sak et al., 2007; Dinçer et al., 2008), Czech (Hajič and Hladká, 1997; Hajič, 2000; Oliva et al., 2000), Slovene (Cussens et al., 1999).
- *Semitic languages*. Arabic (Habash and Rambow, 2005; Zribi et al., 2006), Hebrew (Bar-Haim et al., 2008).
- *Tai languages*. Thai (Ma et al., 2000; Murata et al., 2002; Lu et al., 2003).
- *Other less-studied languages*. Kannada (Vikram and Urs, 2007), Afrikaans (Trushkina, 2007), Telugu (Kumar and Kumar, 2007), Urdu (Anwar et al., 2007), Uyghur (Altenbek, 2006), Kiswahili (Pauw et al., 2006), Vietnamese (Dien and Kiem, 2003), Persian (Mohseni et al., 2008), Bulgarian (Doychinova and Mihov, 2004).

10.6 Conclusion

One of the earliest steps in the processing of natural language text is POS tagging. Usually this is a sentence-based process and given a sentence formed of a sequence of words, we try to assign the correct POS tag to each word. There are basically two difficulties in POS tagging. The first one is the ambiguity in the words, meaning that most of the words in a language have more than one part of speech. The second difficulty arises from the unknown words, the words for which the tagger has no knowledge about. The classical solution to the POS tagging problem is taking the context around the target word into account and selecting the most probable tag for the word by making use of the information provided by the context words.

In this chapter, we surveyed a wide variety of techniques for the POS tagging problem. We can divide these techniques into two broad categories: rule-based methods and statistical methods. The former one was used by the early taggers that attempt to label the words by using a number of linguistic rules. Normally these rules are manually compiled, which is the major drawback of such methods. Later the rule-based systems began to be replaced by statistical systems as sufficient language resources became available. The HMM framework is the most widely used statistical approach for the POS tagging problem. This is probably due to the fact that HMM is a suitable formalism for this problem and it resulted in high success rates in early studies. However, nearly all of the other statistical and machine learning methods are also used to some extent.

POS tagging should not be seen as a theoretical subject. Since tagging is one of the earliest steps in NLP, the results of taggers are being used in a wide range of NLP tasks related to later processing. Probably the most prevalent one is parsing (syntactic analysis) or partial parsing (a kind of analysis limited to particular types of phrases), where the tags of the words in a sentence need to be known in order to determine the correct word combinations (e.g., Pla et al., 2000). Another important application is information extraction, which aims at extracting structured information from unstructured documents. Named-entity recognition, a subtask of information extraction, makes use of tagging and partial parsing in identifying the entities we are interested in and the relationships between these entities (Cardie, 1997).

Information retrieval and question answering systems also make use of the outputs of taggers. The

performance of such systems can be improved if they work on a phrase basis rather than treating each word individually (e.g., Cowie et al., 2000). Finally, we can cite lexical acquisition, machine translation, word-sense disambiguation, and phrase normalization as other research areas that rely on the information provided by taggers.

The state-of-the-art accuracies in POS tagging are around 96%–97% for English-like languages. For languages in other families, similar accuracies are obtained provided that the characteristics of these languages different from English are carefully handled. This seems a quite high accuracy and some researchers argue that POS tagging is an already-solved problem. However, since POS tagging serves as a preprocessing step for higher-level NLP operations, a small improvement has the potential of significantly increasing the quality of later processing. Therefore, we may expect a continuing research effort on this task.

References

- Ahn, Y. and Y. Seo. 2007. Korean part-of-speech tagging using disambiguation rules for ambiguous word and statistical information. In *ICCIT*, pp. 1598–1601, Gyeongju, Republic of Korea. IEEE.
- Alba, E., G. Luque, and L. Araujo. 2006. Natural language tagging with genetic algorithms. *Information Processing Letters* 100(5):173–182.
- Alpaydm, E. 2004. *Introduction to Machine Learning*. MIT Press, Cambridge, MA.
- Altenbek, G. 2006. Automatic morphological tagging of contemporary Uighur corpus. In *IRI*, pp. 557–560, Waikoloa Village, HI. IEEE.
- Altunyurt, L., Z. Orhan, and T. Gngör. 2007. Towards combining rule-based and statistical part of speech tagging in agglutinative languages. *Computer Engineering* 1(1):66–69.
- Anderson, J.M. 1997. *A Notional Theory of Syntactic Categories*. Cambridge University Press, Cambridge, U.K.
- Ando, R.K. 2004. Exploiting unannotated corpora for tagging and chunking. In *ACL*, Barcelona, Spain. ACL.
- Anwar, W., X. Wang, L. Li, and X. Wang. 2007. A statistical based part of speech tagger for Urdu language. In *ICMLC*, pp. 3418–3424, Hong Kong. IEEE.
- Araujo, L. 2002. Part-of-speech tagging with evolutionary algorithms. In *CICLing*, ed. A. Gelbukh, pp. 230–239, Mexico. Springer.
- Asahara, M. and Y. Matsumoto. 2000. Extended models and tools for high-performance part-of-speech tagger. In *COLING*, pp. 21–27, Saarbrücken, Germany. Morgan Kaufmann.
- Bahl, L.R. and R.L. Mercer. 1976. Part-of-speech assignment by a statistical decision algorithm. In *ISIT*, pp. 88–89, Sweden. IEEE.
- Baker, M.C. 2003. *Lexical Categories: Verbs, Nouns, and Adjectives*. Cambridge University Press, Cambridge, U.K.
- Banko, M. and R.C. Moore. 2004. Part of speech tagging in context. In *COLING*, pp. 556–561, Geneva, Switzerland. ACL.
- Bar-Haim, R., K. Sima'an, and Y. Winter. 2008. Part-of-speech tagging of modern Hebrew text. *Natural Language Engineering* 14(2):223–251.
- Bellegarda, J.R. 2008. A novel approach to part-of-speech tagging based on latent analogy. In *ICASSP*, pp. 4685–4688, Las Vegas, NV. IEEE.
- Black, E., F. Jelinek, J. Lafferty, R. Mercer, and S. Roukos. 1992. Decision tree models applied to the labeling of text with parts-of-speech. In *HLT*, pp. 117–121, New York. ACL.

- Boggess, L., J.S. Hamaker, R. Duncan, L. Klimek, Y. Wu, and Y. Zeng. 1999. A comparison of part of speech taggers in the task of changing to a new domain. In *ICLIS*, pp. 574–578, Washington, DC. IEEE.
- Brants, T. 2000. TnT—A statistical part-of-speech tagger. In *ANLP*, pp. 224–231, Seattle, WA.

- Brill, E. 1995a. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics* 21(4):543–565.
- Brill, E. 1995b. Unsupervised learning of disambiguation rules for part of speech tagging. In *Workshop on Very Large Corpora*, eds. D. Yarowsky and K. Church, pp. 1–13, Somerset, NJ. ACL.
- Brill, E. and J. Wu. 1998. Classifier combination for improved lexical disambiguation. In *COLING-ACL*, pp. 191–195, Montreal, QC. ACL/Morgan Kaufmann.
- Cao, H., T. Zhao, S. Li, J. Sun, and C. Zhang. 2005. Chinese post tagging based on bilexical co-occurrences. In *ICMLC*, pp. 3766–3769, Guangzhou, China. IEEE.
- Carberry, S., K. Vijay-Shanker, A. Wilson, and K. Samuel. 2001. Randomized rule selection in transformation-based learning: A comparative study. *Natural Language Engineering* 7(2):99–116.
- Cardie, C. 1997. Empirical methods in information extraction. *AI Magazine* 18(4):65–79.
- Carrasco, R.M. and A. Gelbukh. 2003. Evaluation of TnT tagger for Spanish. In *ENC*, pp. 18–25, Mexico. IEEE.
- Church, K.W. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *ANLP*, pp. 136–143, Austin, TX.
- Clark, A. 2003. Combining distributional and morphological information for part of speech induction. In *EACL*, pp. 59–66, Budapest, Hungary.
- Clark, S., J.R. Curran, and M. Osborne. 2003. Bootstrapping post taggers using unlabelled data. In *CoNLL*, pp. 49–55, Edmonton, AB.
- Collins, M. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *EMNLP*, pp. 1–8, Philadelphia, PA. ACL.
- Cowie, J., E. Ludovik, H. Molina-Salgado, S. Nirenburg, and S. Scheremetyeva. 2000. Automatic question answering. In *RIAO*, Paris, France. ACL.
- Cucerzan, S. and D. Yarowsky. 2000. Language independent, minimally supervised induction of lexical probabilities. In *ACL*, Hong Kong. ACL.
- Curran, J.R. and S. Clark. 2003. Investigating GIS and smoothing for maximum entropy taggers. In *EACL*, pp. 91–98, Budapest, Hungary.
- Curran, J.R., S. Clark, and D. Vadas. 2006. Multi-tagging for lexicalized-grammar parsing. In *COLING/ACL*, pp. 697–704, Sydney, NSW. ACL.
- Cussens, J. 1998. Using prior probabilities and density estimation for relational classification. In *ILP*, pp. 106–115, Madison, WI. Springer.
- Cussens, J., S. Džeroski, and T. Erjavec. 1999. Morphosyntactic tagging of Slovene using Progol. In *ILP*, eds. S. Džeroski and P. Flach, pp. 68–79, Bled, Slovenia. Springer.
- Cutting, D., J. Kupiec, J. Pedersen, and P. Sibun. 1992. A practical part-of-speech tagger. In *ANLP*, pp. 133–140, Trento, Italy.
- Daelemans, W., J. Zavrel, P. Berck, and S. Gillis. 1996. MBT: A memory-based part of speech tagger-generator. In *Workshop on Very Large Corpora*, eds. E. Ejerhed and I. Dagan, pp. 14–27, Copenhagen, Denmark. ACL.
- Darroch, J.N. and D. Ratcliff. 1972. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics* 43(5):1470–1480.
- Dermatas, E. and G. Kokkinakis. 1995. Automatic stochastic tagging of natural language texts. *Computational Linguistics* 21(2):137–163.
- Dickinson, M. and W.D. Meurers. 2003. Detecting errors in part-of-speech annotation. In *EACL*, pp. 107–114, Budapest, Hungary.
- Dien, D. and H. Kiem. 2003. Pos-tagger for English-Vietnamese bilingual corpus. In *HLT-NAACL*, pp. 88–95, Edmonton, AB. ACL.
- Diñçer, T., B. Karaoğlu, and T. Kışla. 2008. A suffix based part-of-speech tagger for Turkish. In *ITNG*,

pp. 680–685, Las Vegas, NV. IEEE.

Doychinova, V. and S. Mihov. 2004. High performance part-of-speech tagging of Bulgarian. In *AIMSA*, eds. C. Bussler and D. Fensel, pp. 246–255, Varna, Bulgaria. Springer.

- Eineborg, M. and N. Lindberg. 2000. ILP in part-of-speech tagging — An overview. In *LLL*, eds. J. Cussens and S. Džeroski, pp. 157–169, Lisbon, Portugal. Springer.
- Finch, A. and E. Sumita. 2007. Phrase-based part-of-speech tagging. In *NLP-KE*, pp. 215–220, Beijing, China. IEEE.
- Florian, R. and G. Ngai. 2001. Multidimensional transformation-based learning. In *CONLL*, pp. 1–8, Toulouse, France. ACL.
- Fossum, V. and S. Abney. 2005. Automatically inducing a part-of-speech tagger by projecting from multiple source languages across aligned corpora. In *IJCNLP*, eds. R. Dale et al., pp. 862–873, Jeju Island, Republic of Korea. Springer.
- Freitag, D. 2004. Toward unsupervised whole-corpus tagging. In *COLING*, pp. 357–363, Geneva, Switzerland. ACL.
- Glass, K. and S. Bangay. 2005. Evaluating parts-of-speech taggers for use in a text-to-scene conversion system. In *SAICSIT*, pp. 20–28, White River, South Africa.
- Grăna, J., G. Andrade, and J. Vilares. 2003. Compilation of constraint-based contextual rules for part-of-speech tagging into finite state transducers. In *CIAA*, eds. J.M. Champarnaud and D. Maurel, pp. 128–137, Santa Barbara, CA. Springer.
- Grzymala-Busse, J.W. and L.J. Old. 1997. A machine learning experiment to determine part of speech from word-endings. In *ISMIS*, pp. 497–506, Charlotte, NC. Springer.
- Habash, N. and O. Rambow. 2005. Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop. In *ACL*, pp. 573–580, Ann Arbor, MI. ACL.
- Hajič, J. 2000. Morphological tagging: Data vs. dictionaries. In *ANLP*, pp. 94–101, Seattle, WA. Morgan Kaufmann.
- Hajič, J. and B. Hladká. 1997. Probabilistic and rule-based tagger of an inflective language — A comparison. In *ANLP*, pp. 111–118, Washington, DC. Morgan Kaufmann.
- Halteren, H.v., J. Zavrel, and W. Daelemans. 2001. Improving accuracy in word class tagging through the combination of machine learning systems. *Computational Linguistics* 27(2):199–229.
- Han, C. and M. Palmer. 2005. A morphological tagger for Korean: Statistical tagging combined with corpus-based morphological rule application. *Machine Translation* 18:275–297.
- Huang, L., Y. Peng, H. Wang, and Z. Wu. 2002. Statistical part-of-speech tagging for classical Chinese. In *TSD*, eds. P. Sojka, I. Kopeček, and K. Pala, pp. 115–122, Brno, Czech Republic. Springer.
- Jiménez, H. and G. Morales. 2002. Sepe: A pos tagger for Spanish. In *CICLing*, ed. A. Gelbukh, pp. 250–259, Mexico. Springer.
- Jung, S., Y.C. Park, K. Choi, and Y. Kim. 1996. Markov random field based English part-of-speech tagging system. In *COLING*, pp. 236–242, Copenhagen, Denmark.
- Kang, M., S. Jung, K. Park, and H. Kwon. 2007. Part-of-speech tagging using word probability based on category patterns. In *CICLing*, ed. A. Gelbukh, pp. 119–130, Mexico. Springer.
- Kempe, A. 1997. Finite state transducers approximating hidden Markov models. In *EACL*, eds. P.R. Cohen and W. Wahlster, pp. 460–467, Madrid, Spain. ACL.
- Kepler, F.N. and M. Finger. 2006. Comparing two Markov methods for part-of-speech tagging of Portuguese. In *IBERAMIA-SBIA*, eds. J.S. Sichman et al., pp. 482–491, Ribeirão Preto, Brazil. Springer.
- Kim, J. and G.C. Kim. 1996. Fuzzy network model for part-of-speech tagging under small training data. *Natural Language Engineering* 2(2):95–110.
- Kim, J., H. Rim, and J. Tsujii. 2003. Self-organizing Markov models and their application to part-of-speech tagging. In *ACL*, pp. 296–302, Sapporo, Japan. ACL.
- Klein, S. and R. Simpson. 1963. A computational approach to grammatical coding of English words. *Journal of ACM* 10(3):334–347.

- Kogut, D.J. 2002. Fuzzy set tagging. In *CICLing*, ed. A. Gelbukh, pp. 260–263, Mexico. Springer.
- Kumar, S.S. and S.A. Kumar. 2007. Parts of speech disambiguation in Telugu. In *ICCIMA*, pp. 125–128, Sivakasi, Tamilnadu, India. IEEE.

- Květon, P. and K. Oliva. 2002a. Achieving an almost correct pos-tagged corpus. In *TSD*, eds. P. Sojka, I. Kopeček, and K. Pala, pp. 19–26, Brno, Czech Republic. Springer.
- Květon, P. and K. Oliva. 2002b. (Semi-)Automatic detection of errors in pos-tagged corpora. In *COLING*, pp. 1–7, Taipei, Taiwan. ACL.
- Lafferty, J., A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, eds. C.E. Brodley and A.P. Danyluk, pp. 282–289, Williamstown, MA. Morgan Kaufmann.
- Lager, T. and J. Nivre. 2001. Part of speech tagging from a logical point of view. In *LACL*, eds. P. de Groote, G. Morrill, and C. Retoré, pp. 212–227, Le Croisic, France. Springer.
- Lee, D. and H. Rim. 2004. Part-of-speech tagging considering surface form for an agglutinative language. In *ACL*, Barcelona, Spain. ACL.
- Lee, G.G., J. Cha, and J. Lee. 2002. Syllable-pattern-based unknown-morpheme segmentation and estimation for hybrid part-of-speech tagging of Korean. *Computational Linguistics* 28(1):53–70.
- Lee, S., J. Tsujii, and H. Rim. 2000a. Part-of-speech tagging based on hidden Markov model assuming joint independence. In *ACL*, pp. 263–269, Hong Kong. ACL.
- Lee, S., J. Tsujii, and H. Rim. 2000b. Hidden Markov model-based Korean part-of-speech tagging considering high agglutinativity, word-spacing, and lexical correlativity. In *ACL*, pp. 384–391, Hong Kong. ACL.
- Lin, H. and C. Yuan. 2002. Chinese part of speech tagging based on maximum entropy method. In *ICMLC*, pp. 1447–1450, Beijing, China. IEEE.
- Lopes, A.d.A. and A. Jorge. 2000. Combining rule-based and case-based learning for iterative part-of-speech tagging. In *EWCBR*, eds. E. Blanzieri and L. Portinale, pp. 26–36, Trento, Italy. Springer.
- Lu, B., Q. Ma, M. Ichikawa, and H. Isahara. 2003. Efficient part-of-speech tagging with a min-max modular neural-network model. *Applied Intelligence* 19:65–81.
- Ma, Q. 2002. Natural language processing with neural networks. In *LEC*, pp. 45–56, Hyderabad, India. IEEE.
- Ma, Q., M. Murata, K. Uchimoto, and H. Isahara. 2000. Hybrid neuro and rule-based part of speech taggers. In *COLING*, pp. 509–515, Saarbrücken, Germany. Morgan Kaufmann.
- Manning, C.D. and H. Schütze. 2002. *Foundations of Statistical Natural Language Processing*. 5th ed., MIT Press, Cambridge, MA.
- Maragoudakis, M., T. Ganchev, and N. Fakotakis. 2004. Bayesian reinforcement for a probabilistic neural net part-of-speech tagger. In *TSD*, eds. P. Sojka, I. Kopeček, and K. Pala, pp. 137–145, Brno, Czech Republic. Springer.
- Marcus, M.P., B. Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics* 19(2):313–330.
- Marques, N.C. and G.P. Lopes. 2001. Tagging with small training corpora. In *IDA*, eds. F. Hoffmann, D.J. Hand, N.M. Adams, D.H. Fisher, and G. Guimaraes, pp. 63–72, Cascais, Lisbon. Springer.
- Màrquez, L., L. Padró, and H. Rodríguez. 2000. A machine learning approach to pos tagging. *Machine Learning* 39:59–91.
- Mayfield, J., P. McNamee, C. Piatko, and C. Pearce. 2003. Lattice-based tagging using support vector machines. In *CIKM*, pp. 303–308, New Orleans, LA. ACM.
- McCallum, A., D. Freitag, and F. Pereira. 2000. Maximum entropy Markov models for information extraction and segmentation. In *ICML*, pp. 591–598, Stanford, CA. Morgan Kaufmann.
- Merialdo, B. 1994. Tagging English text with a probabilistic model. *Computational Linguistics* 20(2): 155–171.
- Mihalcea, R. 2003. Performance analysis of a part of speech tagging task. In *CICLing*, ed. A. Gelbukh, pp. 158–167, Mexico. Springer.

Mikheev, A. 1997. Automatic rule induction for unknown-word guessing. *Computational Linguistics* 23(3):405–423.

- Mohammed, S. and T. Pedersen. 2003. Guaranteed pre-tagging for the Brill tagger. In *CICLing*, ed. A. Gelbukh, pp. 148–157, Mexico. Springer.
- Mohseni, M., H. Motalebi, B. Minaei-bidgoli, and M. Shokrollahi-far. 2008. A Farsi part-of-speech tagger based on Markov model. In *SAC*, pp. 1588–1589, Ceará, Brazil. ACM.
- Mora, G.G. and J.A.S. Peiró. 2007. Part-of-speech tagging based on machine translation techniques. In *lbPRIA*, eds. J. Martí et al., pp. 257–264, Girona, Spain. Springer.
- Murata, M., Q. Ma, and H. Isahara. 2002. Comparison of three machine-learning methods for Thai part-of-speech tagging. *ACM Transactions on Asian Language Information Processing* 1(2):145–158.
- Nagata, M. 1999. A part of speech estimation method for Japanese unknown words using a statistical model of morphology and context. In *ACL*, pp. 277–284, College Park, MD.
- Nakagawa, T., T. Kudo, and Y. Matsumoto. 2002. Revision learning and its application to part-of-speech tagging. In *ACL*, pp. 497–504, Philadelphia, PA. ACL.
- Nakagawa, T. and Y. Matsumoto. 2006. Guessing parts-of-speech of unknown words using global information. In *CL-ACL*, pp. 705–712, Sydney, NSW. ACL.
- Nasr, A., F. Béchet, and A. Volanschi. 2004. Tagging with hidden Markov models using ambiguous tags. In *COLING*, pp. 569–575, Geneva. ACL.
- Ning, H., H. Yang, and Z. Li. 2007. A method integrating rule and HMM for Chinese part-of-speech tagging. In *ICIEA*, pp. 723–725, Harbin, China. IEEE.
- Oliva, K., M. Hnátková, V. Petkevič, and P. Květon. 2000. The linguistic basis of a rule-based tagger for Czech. In *TSD*, eds. P. Sojka, I. Kopeček, and K. Pala, pp. 3–8, Brno, Czech Republic. Springer.
- Padró, L. 1996. Posttagging using relaxation labelling. In *COLING*, pp. 877–882, Copenhagen, Denmark.
- Pauw, G., G. Schyver, and W. Wagacha. 2006. Data-driven part-of-speech tagging of Kiswahili. In *TSD*, eds. P. Sojka, I. Kopeček, and K. Pala, pp. 197–204, Brno, Czech Republic. Springer.
- Pérez-Ortiz, J.A. and M.L. Forcada. 2001. Part-of-speech tagging with recurrent neural networks. In *IJCNN*, pp. 1588–1592, Washington, DC. IEEE.
- Peshkin, L., A. Pfeffer, and V. Savova. 2003. Bayesian nets in syntactic categorization of novel words. In *HLT-NAACL*, pp. 79–81, Edmonton, AB. ACL.
- Pla, F. and A. Molina. 2004. Improving part-of-speech tagging using lexicalized HMMs. *Natural Language Engineering* 10(2):167–189.
- Pla, F., A. Molina, and N. Prieto. 2000. Tagging and chunking with bigrams. In *COLING*, pp. 614–620, Saarbrücken, Germany. ACL.
- Poel, M., L. Stegeman, and R. op den Akker. 2007. A support vector machine approach to Dutch part-of-speech tagging. In *IDA*, eds. M.R. Berthold, J. Shawe-Taylor, and N. Lavrač, pp. 274–283, Ljubljana, Slovenia. Springer.
- Portnoy, D. and P. Bock. 2007. Automatic extraction of the multiple semantic and syntactic categories of words. In *AIAP*, pp. 514–519, Innsbruck, Austria.
- Prins, R. 2004. Beyond n in n-gram tagging. In *ACL*, Barcelona, Spain. ACL.
- Pustet, R. 2003. *Copulas: Universals in the Categorization of the Lexicon*. Oxford University Press, Oxford, U.K.
- Raju, S.B., P.V.S. Chandrasekhar, and M.K. Prasad. 2002. Application of multilayer perceptron network for tagging parts-of-speech. In *LEC*, pp. 57–63, Hyderabad, India. IEEE.
- Ramshaw, L.A. and M.P. Marcus. 1994. Exploring the statistical derivation of transformation rule sequences for part-of-speech tagging. In *ACL*, pp. 86–95, Las Cruces, NM. ACL/Morgan Kaufmann.
- Rapp, R. 2005. A practical solution to the problem of automatic part-of-speech induction from text. In *ACL*, pp. 77–80, Ann Arbor, MI. ACL.
- Ratnaparkhi, A. 1996. A maximum entropy model for part-of-speech tagging. In *EMNLP*, eds. E. Brill

and K. Church, pp. 133–142, Philadelphia, PA. ACL.

Ratnaparkhi, A. 1998. Maximum entropy models for natural language ambiguity resolution. PhD dissertation, University of Pennsylvania, Philadelphia, PA.

- Reiser, P.G.K. and P.J. Riddle. 2001. Scaling up inductive logic programming: An evolutionary wrapper approach. *Applied Intelligence* 15:181–197.
- Reynolds, S.M. and J.A. Bilmes. 2005. Part-of-speech tagging using virtual evidence and negative training.
In *HLT-EMNLP*, pp. 459–466, Vancouver, BC. ACL.
- Roche, E. and Y. Schabes. 1995. Deterministic part-of-speech tagging with finite-state transducers.
Computational Linguistics 21(2):227–253.
- Rocio, V., J. Silva, and G. Lopes. 2007. Detection of strange and wrong automatic part-of-speech tagging.
In *EPIA*, eds. J. Neves, M. Santos, and J. Machado, pp. 683–690, Guimarães, Portugal. Springer.
- Roth, D. and D. Zelenko. 1998. Part of speech tagging using a network of linear separators. In *COLING-ACL*, pp. 1136–1142, Montreal, QC. ACL/Morgan Kaufmann.
- Sak, H., T. Güngör, and M. Saraçlar. 2007. Morphological disambiguation of Turkish text with perceptron algorithm. In *CICLing*, ed. A. Gelbukh, pp. 107–118, Mexico. Springer.
- Schmid, H. 1994. Part-of-speech tagging with neural networks. In *COLING*, pp. 172–176, Kyoto, Japan. ACL.
- Schütze, H. 1993. Part-of-speech induction from scratch. In *ACL*, pp. 251–258, Columbus, OH. ACL.
- Schütze, H. 1995. Distributional part-of-speech tagging. In *EACL*, pp. 141–148, Belfield, Dublin. Morgan Kaufmann.
- Schütze, H. and Y. Singer. 1994. Part-of-speech tagging using a variable memory Markov model. In *ACL*, pp. 181–187, Las Cruces, NM. ACL/Morgan Kaufmann.
- Sun, M., D. Xu, B.K. Tsou, and H. Lu. 2006. An integrated approach to Chinese word segmentation and part-of-speech tagging. In *ICCPOL*, eds. Y. Matsumoto et al., pp. 299–309, Singapore. Springer.
- Sündermann, D. and H. Ney. 2003. Synther — A new m-gram postagger. In *NLP-KE*, pp. 622–627, Beijing, China. IEEE.
- Tapanainen, P. and A. Voutilainen. 1994. Tagging accurately — Don't guess if you know.
In *ANLP*, pp. 47–52, Stuttgart, Germany.
- Taylor, J.R. 2003. *Linguistic Categorization*. 3rd ed., Oxford University Press, Oxford, U.K.
- Thede, S.M. 1998. Predicting part-of-speech information about unknown words using statistical methods.
In *COLING-ACL*, pp. 1505–1507, Montreal, QC. ACM/Morgan Kaufmann.
- Thede, S.M. and M.P. Harper. 1999. A second-order hidden Markov model for part-of-speech tagging.
In *ACL*, pp. 175–182, College Park, MD. ACL.
- Toutanova, K., D. Klein, C.D. Manning, and Y. Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT-NAACL*, pp. 252–259, Edmonton, AB. ACL.
- Toutanova, K. and C.D. Manning. 2000. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *EMNLP/VLC*, pp. 63–70, Hong Kong.

- Triviño-Rodríguez, J.L. and R. Morales-Bueno. 2001. Using multiattribute prediction suffix graphs for Spanish part-of-speech tagging. In *IDA*, eds. F. Hoffmann et al., pp. 228–237, Lisbon, Portugal. Springer.
- Trushkina, J. 2007. Development of a multilingual parallel corpus and a part-of-speech tagger for Afrikaans. In *IIP III*, eds. Z. Shi, K. Shimohara, and D. Feng, pp. 453–462, New York, Springer.
- Tsuruoka, Y., Y. Tateishi, J. Kim et al. 2005. Developing a robust part-of-speech tagger for biomedical text. In *PCI*, eds. P. Bozanis and E.N. Houstis, pp. 382–392, Volos, Greece. Springer.
- Tsuruoka, Y. and J. Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *HLT/EMNLP*, pp. 467–474, Vancouver, BC. ACL.
- Tür, G. and K. Oflazer. 1998. Tagging English by path voting constraints. In *COLING*, pp. 1277–1281, Montreal, QC. ACL.
- Vikram, T.N. and S.R. Urs. 2007. Development of prototype morphological analyzer for the south Indian language of Kannada. In *ICADL*, eds. D.H.-L. Goh et al., pp. 109–116, Hanoi, Vietnam. Springer.
- Villamil, E.S., M.L. Forcada, and R.C. Carrasco. 2004. Unsupervised training of a finite-state sliding- window part-of-speech tagger. In *ESTAL*, eds. J.L. Vicedo et al., pp. 454–463, Alicante, Spain. Springer.

1. Chen J, Benesty J, Huang YA (2006) Time delay estimation in room acoustic environments: an overview. *EURASIP J Adv Signal Process* 2006:1–1
2. Cobos M, Marti A, Lopez JJ (2011) A modified SRP-PHAT functional for robust real-time sound source localization with scalable spatial sampling. *IEEE Signal Process Lett* 18(1):71–74
3. Deleforge A, Horaud, R (2012) The cocktail party robot: sound source separation and localisation with an active binaural head. In: *Proceedings 7th ACM/IEEE international conference human-Robot interaction (HRI)*, pp 431–438
4. Deller JR, Proakis JG, Hansen JHL (2000) *Discrete-time processing of speech signals*. Wiley-IEEE Press, New York
5. DiBiase JH (2000) A high accuracy, low-latency technique for talker localization in reverberant environments using microphone arrays. PhD thesis, Brown University
6. DiBiase JH, Silverman HF, Brandstein MS (2001) Robust localization in reverberant rooms. *Microphone arrays: signal processing techniques and applications*, pp 157–180
7. Dmochowski J, Benesty J, Affes S (2007) Direction of arrival estimation using the parameterized spatial correlation matrix. *IEEE Trans Audio, Speech, Lang Process* 15(4):1327–1339
8. Dmochowski J, Benesty J, Affes S (2007) A generalized steered response power method for computationally viable source localization. *IEEE Trans Audio, Speech, Lang Process* 15(8):2510–2526
9. Dmochowski J, Benesty J, Affes S (2008) Linearly constrained minimum variance source localization and spectral estimation. *IEEE Trans Audio, Speech, Lang Process* 16(8):1490–1502
10. Do H, Silverman HF, Yu Y (2007) A real-time SRP-PHAT source location implementation using stochastic region contraction (SRC) on a large-aperture microphone array. In: *Proceedings IEEE international conference on acoustics, speech and signal processing (ICASSP'07)*, vol 1, pp 1–121–1–124
11. Fallon MF, Godsill S (2010) Acoustic source localization and tracking using track before detect. *IEEE Trans Audio, Speech, Lang Process* 18(6):1228–1242
12. Fallon MF, Godsill S (2012) Acoustic source localization and tracking of a time-varying number of speakers. *IEEE Trans Audio, Speech, Lang Process* 20(4):1409–1415
13. Gannot S, Dvorkind TG (2006) Microphone array speaker localizers using spatial-temporal information. *EURASIP J Appl Signal Process (special issue on microphone arrays)* 2006:1–17
14. Garofolo J, Lamel L, Fisher W, Fiscus J, Pallett D, Dahlgren N, Zue V (1993) *TIMIT acoustic-phonetic continuous speech corpus*. Philadelphia
15. Gordon NJ, Salmond DJ, Smith AFM (1993) Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In: *Proceedings of IEE -F, radar and signal processing*, vol 140, pp 107–113. IET
16. Grewal MS, Andrews AP (2011) *Kalman filtering: theory and practice using MATLAB*. Wiley, New York
17. Griffin DW, Lim JS (1988) Multiband excitation vocoder. *IEEE Trans Acoust Speech Signal Process* 36(8):1223–1235
18. Habets EAP, Benesty J (2012) A perspective on frequency-domain beamformers in room acoustics. *IEEE Trans Audio, Speech, Lang Process* 20(3):947–960
19. Habets EAP, Benesty J, Naylor PA (2012) A speech distortion and interference rejection constraint beamformer. *IEEE Trans Audio, Speech, Lang Process* 20(3):854–867
20. Huang Y, Benesty J, Elko GW, Mersereau RM (2001) Real-time passive source localization: a practical linear-correction least-squares approach. *IEEE Trans Speech, Audio Process* 9(8):943–956
21. Johnson DH, Dudgeon DE (1992) *Array signal processing: concepts and techniques*. Simon & Schuster
22. Knapp CH, Carter GC (1976) The generalized correlation method for estimation of time delay.

IEEE Trans Acoust Speech Signal Process 24(4):320–327

23. Lehmann EA, Johansson AM (2007) Particle filter with integrated voice activity detection for acoustic source tracking. EURASIP J Adv Signal Process 2007

24. Lehmann EA, Johansson AM (2008) Prediction of energy decay in room impulse responses simulated with an image-source model. *J Acoust Soc Am* 124(1):269–277
25. Levy A, Gannot S, Habets EAP (2011) Multiple-hypothesis extended particle filter for acoustic source localization in reverberant environments. *IEEE Trans Audio, Speech, Lang Process* 19(6):1540–1555
26. Ma W-K, VoB-N, Singh SS, Baddeley A (2006) Tracking an unknown time-varying number of speakers using TDOA measurements: a random finite set approach. *IEEE Trans Signal Process* 54(9):3291–3304
27. Marti A, Cobos M, Lopez JJ (2011) Real time speaker localization and detection system for camera steering in multiparticipant videoconferencing environments. In: *Proceedings of IEEE international conference on acoustics, speech, signal processing (ICASSP'11)*, pp 2592–2595
28. Morelande MR, Kreucher CM, Kastella K (2007) A bayesian approach to multiple target detection and tracking. *IEEE Trans Signal Process* 55(5):1589–1604
29. Talantzis F (2010) An acoustic source localization and tracking framework using particle filtering and information theory. *IEEE Trans Audio, Speech, Lang Process* 18(7):1806–1817
30. Timofeev S, Bahai ARS, Varaiya P (2008) Adaptive acoustic beamformer with source tracking capabilities. *IEEE Trans Signal Process* 56(7):2812–2820
31. Varga A, Steeneken HJM (1993) Assessment for automatic speech recognition: II. NOISEX- 92: a database and an experiment to study the effect of additive noise on speech recognition systems. *Speech Commun* 12(3):247–251
32. Van Veen BD, Buckley KM (1988) Beamforming: a versatile approach to spatial filtering. *IEEE ASSP Magazine* 5(2):4–24
33. Vermaak J, Blake A (2001) Nonlinear filtering for speaker tracking in noisy and reverberant environments. In: *Proceedings of IEEE International Conference on Acoustics, Speech, Signal Processing (ICASSP'01)*, pp 3021–3024
34. VoB-T, VoB-N, Antonio C (2008) Bayesian filtering with random finite set observations. *IEEE Trans Signal Process* 56(4):1313–1326
35. Ward DB, Lehmann EA, Williamson RC (2003) Particle filtering algorithms for tracking an acoustic source in a reverberant environment. *IEEE Trans Speech Audio Process* 11(6):826–836
36. Wu K, Goh ST, Khong AWH (2013) Speaker localization and tracking in the presence of sound interference by exploiting speech harmonicity. In: *Proceedings of IEEE international conference on acoustics, speech, signal processing (ICASSP'13)*, pp 365–369
37. Zeng W-J, Li X-L (2010) High-resolution multiple wideband and nonstationary source localization with unknown number of sources. *IEEE Trans Signal Process* 58(6):3125–3136

6.Related Work & Discussion

2.1 TEMPLATE MATCHING

2.1.1 ELIZA

In 1950, Alan Turing proposed that a machine should be considered intelligent if a human being, communicating with it by teletype, could not distinguish it from another human being. In 1966, Joseph Weizenbaum published a computer program called ELIZA that seemed to pass Turing's test (admittedly with a different kind of conversation than Turing envisioned) but obviously did not contain humanlike intelligence.

ELIZA carried on a dialogue with its user, in English, in the manner of a psycho-logical counselor. Figure 2.1 shows a typical dialogue (from Weizenbaum 1966). ELIZA was so convincing that people willingly told it their troubles and asked Weizenbaum to go away so they could converse in private. Psychiatrists even thought it was a practical way of administering therapy. Weizenbaum was alarmed at this; far from a triumph of artificial intelligence, he considered ELIZA to be evidence of how easy it is to deceive people with computers.

After all, ELIZA did not “understand” much of its input, if “understanding” means responding to the information content. ELIZA merely recognized certain patterns of words and gave appropriate “canned” responses. For example, if the user typed *Yoi are X*, ELIZA could respond *I ml makes you think I rim X*, where N is any adjective. Any

Men are all alike.
IN WHAT WAY

They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED

It's true. I am unhappy.
DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY

I need some help, that much seems certain.
WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP

Perhaps I could learn to get along with my mother.
TELL ME MORE ABOUT YOUR FAMILY

My mother takes care of me.
WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU

My father.
YOUR FATHER

You are like my father in some ways.
WHAT RESEMBLANCE DO YOU SEE

You are not very aggressive but 'I think you don't want me to
notice that.

WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE

You don't argue with me.

WHY DO YOU THINK I DON'T ARGUE WITH YOU

You are afraid of me.

DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU
My father is afraid of everybody.
WHAT ELSE COMES TO MIND WHEN YOU THINK OF YOUR FATHER
Bullies.
DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR
BOYFRIEND MADE YOU COME HERE

Figure 2.1 A sample session with ELIZA. The machine's responses are in upper case.

sentence containing words like *mother* would get a reply such as *Tell me more about your family*. And a sentence containing nothing that ELIZA recognized would evoke a response such as *Tell me more* or *Go on*.

Exercise 2.1.1.1

If your computer has the Gnu Einacs editor, you can play with a somewhat updated version of ELIZA. Simply get into Emacs and type: *Esc x dock* or *Return*.

2.1.2 Other Template Systems

ELIZA was an example of a TEMPLATE SYSTEM or PATTERN-MATCHING SYSTEM, a tech-

dialogue.

Several other classic AI programs used templates. One of these was SIR (Semantic Information Retrieval) by Bertram Raphael (1968). SIR could do enough logical reasoning to answer questions such as, “Every person has two hands. Every hand has two fingers. Joe is a person. How many fingers does Joe have?”

Another was Bobrow’s STUDENT, which could solve high—school-level mathematical problems stated in English. A noteworthy feature of STUDENT is that it used recursive templates. For example, the template *if X then Y* allowed whole sentences in place of X and F and would apply whole-sentence templates to them. Bobrow (1968) describes the original system and Norvig (1991) reimplements it in Common Lisp.

SIR, STUDENT, and their kin were developed mainly to study the information content of natural language, not to model the syntax. Nobody claims that templates are a realistic model of the way the human mind processes language. They are, however, a quick way to extract useful information from natural-language input, adequate for many practical user-interface applications. One modern example is HAL, the English-language command interface for the Lotus 1-2-3 spreadsheet program. Though its internal workings have not been made public, HAL shows the rigidity of syntax that is typical of a template system (see Petzold 1987).

!2 DOS COMMANDS IN ENGLISH

2.2.1 Recipe for a Template System

In this section we will develop a template system that lets the user type commands to MS—DOS (PC—DOS) in English. The same could be done just as easily for UNIX or another operating system. See Lane (1987) for a different DOS—in-English system written in Turbo Prolog.

Like most template systems, ours will translate English into a formal language, in this case DOS command language. The formal language has two important characteristics:

- It is much less expressive than English; many English words and expressions simply cannot be translated into it.
- The human user knows about these restrictions on expressive power and can restrict the English input accordingly. For example, people know that when talking to DOS, it makes sense to say *Show me the BAT files on drive D* but not *Look on my works, ye mi ghy, and despair*.

This means that the template system can get away with covering only a tiny part of the English language.

The first step in designing a template system is to write down a lot of sample input sentences with the desired translations, then try to find patterns in them. For example:

<i>What files are on drive B?</i>	
<i>Delete all my files.</i>	<code>dir b:</code>
<i>Run the word processor.</i>	<code>erase *.*</code>
	$\Rightarrow$ wp

Figure 2.2 shows a larger set.

At this stage you may well decide that DOS command language isn't powerful enough. For example, the user can't say *Erase all files older than 1729790* because there is no DOS command corresponding to it; the *erase* command does not check the date of a file. This is a separate issue from natural language processing; we'll leave it behind, but you have been warned. The success of any natural—language—driven software depends on whether the software can do what the user wants, not just whether it understands English.

The next step is to write a set of rules to do the translation. These will be of two

types:

- SIMPLIFICATION RuLEs discard unnecessary words and make equivalent words look alike. For example, *display* might get rewritten as *show*, and *the* might get dropped altogether.
- ANSLATION RuLEs actually map a template into the formal language, such as `[show, files, on, disk, X] ['dir', X, ':']`.

<i>What is on disk A?</i>	$\Rightarrow$ <code>dir a:</code>

<i>What is on the disk in drive A?</i>	=> <code>dir a:</code>
<i>What files are on disk A?</i>	<code>dir a:</code>
<i>What files are there on the disk in drive A?</i> (Likewise for any drive.)	=> <code>dir a:</code>
<i>Are there any EXE files on disk A?</i>	<code>d'xr a:*.exe</code>
<i>Are there any EXE files on the disk in drive A?</i>	<code>dir a:*.exe</code>
<i>What EXE files are there on disk A?</i>	<code>dir a:*.exe</code>
<i>What EXE files are there on the disk in drive A?</i> (Likewise for BAT files, COM files, etc.)	<code>dir a:*.exe</code>
<i>Copy everything from disk A to disk B.</i>	<code>copy a:*. * b:</code>
<i>Copy everything from (the disk in)</i> <i>drive A to (the disk in) drive B.</i>	<code>copy a:*. * b:</code>
<i>Copy (all) files from . . . (etc.)</i>	=> <code>copy a: * * b:</code>
<i>Copy all BAT files from . . . (etc.)</i>	<code>copy a:*.bat b:</code>

Figure 2.2 Some English sentences and their translations into DOS command language.

<i>are</i>	$\Rightarrow$	B
<i>there</i>		
	$\Rightarrow$	$\emptyset$
<i>disk in drive</i>		<i>drive</i>
<i>disk in</i>		<i>drive</i>
<i>disk</i>		<i>drive</i>
<i>what files</i>		<i>files</i>
<i>what</i>		<i>files</i>
<i>file</i>		<i>files</i>
<i>everything</i>		<i>all files</i>

Translation rules:

<i>Files on drive X?</i>	dir X :
<i>X files on drive Y?</i>	dir Y : * . X
<i>Copy all files from drive X to drive Y.</i>	copt X : * * Y :
<i>Copy all X files from drive Y to drive Z.</i>	copt Y : * . X Z :

Figure 2.3 Examples of simplification and translation rules.

The purpose of the simplification rules is to reduce the number of translation rules. There's a trade-off; if you put in too many simplification rules, you may lose distinctions that you later on want to preserve. It's common for some of the simplification rules to get dropped as the system becomes more complete.

Figure 2.3 shows simplification and translation rules to handle the sentences in Figure 2.2. Note that every sentence goes through all the simplification rules that match any part of it, and then through *one* translation rule (the one that matches the whole sentence).

What about sentences that the system can't recognize? Some of these will be typing errors and the like. Others will be reasonable sentences that you didn't provide for. The appropriate thing to do is store them on a file so that you can come back and modify the system later to handle them.

Exercise 2.2.1.1

In addition to the examples in Figure 2.2, write down at least 20 more English sentences and their translations into MS-DOS command language (or the command language of your computer).

Exercise 2.2.1.2

Write a set of simplification and translation rules (in any notation you care to use) to handle those sentences. Notice that if you do this intelligently, your rules will also handle many sentences that you didn't originally think of.

2.2.2 Implementing Simplification Rules

Simplification rules use templates for *parts* of sentences. For example, the word *everything* gets simplified to *all files* wherever it occurs, and the phrase *what files* is simplified to *ales*.

What we need is a way to match simplification rules to the input list. The match can occur anywhere in the list. Accordingly, the program works through the list, word by word, comparing it to all the templates at every point:

```
[copy, files, from, disk, in, drive, a, to, driv (No match)
                                     e, b] [No match)
[files, from, disk, in, drive, a, co, driv [No match)
                                     e, b]
(from, disk, in, drive, a, to, drive, b] Matches 'disk in')
[disk, in, drive, a, to, drive, b]
```

The next question is how to store the templates. Here open lists come in handy. It's much easier to match [di s k , in , drive , a , C o , dr ive , b) with di st , in X] than with [di sk, in) . The latter would require a special procedure to compare elements one by one; the former can be done by ordinary Prolog unification. Here, then, are a set of simplification rules expressed as open lists:

```
% sr(T1,T2)
% Simplification rules.
```



```

sr([thelX],X).
sr([islX],X).
sr([arelX],X).
sr([therelX1,X).
sr([anylX],X).
sr([disk,in,drivelX],[drivelX]).
sr([disk,inIXJ,[drivelX])
srl[disklX],[drivelX]).
srl[what,fileslX],[fileslX]).
sr([whatlX],[fileslX]).

sr([filelX],[fileslX]).
sr([everythingIX],[all,fileslX]).
sr([everylX],[all(X)].

```

That is: [the !X simplifies to X; [di sk , in 1 X) simplifies to [dr ive X; and so

The order of the simplification rules matters because more than one rule can match

and

disk drive.

If both rules match, the first one should take precedence; thus it must be tried first. Otherwise the result might be *drive in drive A*.

After one simplification rule works on the string, the result gets fed into the simplification rules again. This makes it possible, for example, to simplify *all files* to *every file* and then change *every file* to *everything*. You can even get loops; a loop will happen if there is a rule that changes *every* to *all* and also a rule that changes *all* to *every*. The complete simplification routine looks like this:

```
% simplify(+List,-Result)
% Applies simplification rules to List giving Result.

simplify(List,Result) :-                % A simp. rule matches
    sr(List,NewList),                  % so apply it and then
                                     % try to further simplify
    simplify(NewList,Result).           % the result

simplify([WlWords],[WlNewWords]) :-    % No simp. rule matches
    simplify(Words,NewWords).           % so advance to next word

simplify([],[]).                        % No more words
```

Exercise 2.2.2.1

Using `simplify/2` given above, implement all the simplification rules that you developed in Exercise 2.2.1.2.

2.2.3 Implementing Translation Rules

Translation rules are relatively simple because each of them is supposed to match the whole list of words:

```
% tr(?Input,?Result)

% Translation rules.

tr([quit],[quit]).

tr([files,on,drive,X,'?'J,['dir ',X,':'])).

tr([X,files,on,drive,Y,'?'],['dir ',Y,':*.',X]).
tr([copy,all,files,from,drive,X,to,drive,Y,'],
    ['COpy ',X,':*.' ' <'1'] )
```

The output of a translation rule is a list of atoms which, when converted back into character strings and concatenated, will give the appropriate DOS command. (The first of these rules handles the “quit” command that the user will use to exit from the program.)

The procedure that applies the translation rules will simply find a rule that applies to the input, then execute a cut, or complain if no rule is applicable:

```
% translate(-Input,+Result)
% Applies a translation rule, or complains
% if no translation rule matches the input.

translate(Input,Result) -
    tr(Input,Result),

translated , [])

write('I do not understand. '),
nl.
```

Exercise 2.2.3.1

Implement all the translation rules that you developed in Exercise 2.2.1.2. Make sure these fit together properly with the simplification rules that you have just implemented.

2.2.4 The Rest of the System

Once the translation rules have produced a command, the next step is to pass the command to the operating system. How this is done depends on the exact version of Prolog.

The first step in ALS Prolog (which is typical) is to concatenate the atoms into a character string, so that for example [' dir , o :] becomes "dir a : ". This is simple; just expand each atom into a list of characters, and append the lists:

```
% make_string(+ListOfAtoms,-String)
% Concatenates a list of atoms giving a string.
% Example: ['a','b','c'] gives "abc".

make_string([HlT],Result)
    name(H,Hstring),
    make_string(T,Tstring),
    append(Hstring,Tstring,Result).

make_string([],[]).

% append(°List1,?List2,°List3)
% Appending List1 to List2 gives List3.
```

```
% pass_to_os(+Command)

%   Accepts a command as a list of atoms, concatenates
%   the atoms, and passes the command to the operating system.

pass_to_os([quit])

pass_to_os([]) :- !.

pass_to_os(Command)
    make_string(Command,S),system(S).
```

Notice that `pass_to_os` ignores `[quit]` (the quit command) and `[]` (the output of an unrecognized command).'

Finally, we're ready to define the main procedure of the template system. Its purpose is to accept sentences from the user and run them through `simplify`, `translatace`, and `pass_commands` in succession. The main loop uses `repeat` so that execution will continue even if the processing of any particular command fails. Here, then, is the main procedure:

```
% process commands

%   Repeatedly accepts commands in English,
%   simplifies and translates them,
%   and passes them to the operating system.

process_commands
    repeat,

        write('Command: '),

        read_atomics(Words),      % defined in Appendix B
        simplify(Words,SimplifiedWords),
        translate(SimplifiedWords,Command),
        pass_to_os(Command),

        Command == [quit],
```

The template system is now complete.

'The equivalent of `make_string(Command, s), system(S)` in some other Prologs is:

Arity Prolog: LPA	<code>concat(CommaDd, S), shell(S)</code>
(MS-DOS):	<code>make_string(Command, S), name(C, S), dos(C)</code>
ESL Prolog—2:	<code>make_string(Cowand, S), list(S, C), command(C, 32000, display)</code>
Quintus (UNIX):	<code>make_string(Cowand, S), name(C, S), unix(system(C))</code>
Quintus (VMS):	<code>make_string(Connand, S), name(C, S), vns(dcl(C))</code>

Consult your manual *Not* further guidance.

Exercise 2.2.4.1

Find out how your Prolog passes commands to the operating system, and get `pass to os` working. Verify that

```
?- pass to os([dir]).
```

(or the equivalent) works correctly.

Exercise 2.2.4.2

Modify `pass to os` so that it displays the command on the screen before passing it to the operating system.

Exercise 2.2.4.3

Using your answers to several previous exercises, build a working, though small, template system that translates English sentences into DOS commands.

Exercise 2.2.4.4

Modify your DOS-in-English system so that if the user types an unrecognized sentence, this sentence is recorded in a file for you to see. (This will enable you to improve the system later, to accommodate more of the sentences that users actually type.)

[Hint: Consult your Prolog manual to find out how to open a file in append mode (i.e., open an already existing file so that you can write at the end).]

Exercise 2.2.4.5 (project)

Expand your DOS-in-English system so that it becomes a practical (if still modest) piece of software. Modify it in any ways you can think of that will make it more useful. This is a very open-ended project; depending on how much work you choose to do, it could be anything from a weekend assignment to a Ph.D. thesis.

Exercise 2.2.4.6 (project)

ELIZA in Prolog. Implement a version of ELIZA. To do this, you will have to implement:

- A simplifier that can translate you to *me* and mc to you without getting into a loop;
- A matching algorithm more powerful than ordinary Prolog unification, so that a variable can match more than one word in a list (for example, X, s acs , LhaL , Y] should match *my mother snys that ciirplanes are dangerous*),
- A strategy for identifying the most important word in a sentence (for example, *computer* is more important than *mother*, which is more important than *why*), so that if more than one template matches the input, the template that recognizes the most important word can be chosen;
- A way to give different responses to the same question at different times.

Weizenbaum (1966) gives a complete, though disorganized, set of simplification and trans-

English grammar and style checking. Some commonly misused words and phrases can be recognized by a template system. Examples:

- *comprised of* (standard English is *composed of*),
 - *inasmuch as*, *in point of fact*, *in the area of* (these are grammatical but should usually be replaced by clearer, more direct wordings such as *because*, *in fact*, or *in*);
 - *very* (much overused; should often be deleted);
- n does not only* (a common mistake in the English of people whose native language is German or Scandinavian).

Write a program that will read English documents (business letters, computer documentation, etc.), and point out words and phrases such as these.

1.3 KEYWORD ANALYSIS

2.3.1 A Time-Honored Approach

An alternative to template matching is KEYWORD ANALYSIS. Instead of matching the whole sentence to a template, a keyword system looks for specific words in the sentence and responds to each word in a specific way.

One of the first keyword systems was that of Blum (1966). Inspired by ELIZA, Blum wrote a program to accept sentences such as

Copy 1 file from U1 to U2, binary, 556 bpi.

and convert them into commands for a program that managed magnetic tapes.

Though he took ELIZA as his model, Blum ended up using quite different techniques. He distinguished three kinds of keywords: requests (*list*, *copy*, *backspace*, etc.), “qualifiers” that further described the request, such as *binary* or *blocked*, and “quantifiers” (numbers) such as *5 files* or *556 bpi* (= bits per inch). His program simply collected all the requests, qualifiers, and quantifiers in the

input and put them together into a properly formed command, paying very little attention to the word order and completely ignoring unrecognized words.

Keyword analysis has been reinvented several times, with slight variations, and keyword systems have had a long and successful history. Unlike template systems, they are not thrown off by slight variations of wording; unrecognized words are simply skipped. Keyword systems are useful in any situation where the input is known to contain certain kinds of information, and other information, if any, can be ignored.

Two prominent keyword systems today are AICorp's Intellect and Symantec's Q&A; Obermeier (1989) lists many others. Both Q&A and Intellect are database query systems and both work very much like the system to be developed later in this chapter. Wallace (1984) describes a much more elaborate keyword system written in Prolog.

2.3.2 Database Querying

In this section we will develop a keyword-analysis program to process database queries that are expressed in English. Before doing this, however, we must say a few things about databases.

Tables and tuples. A RELATIONAL DATABASE consists of one or more TABLES such as the following:

ID number	Name	Birth date	Title	Salary
1001	Doe, John P.	1947/01/30	President	100000
1002	Smith, Mary J.	1960/09/05	Programmer	30000
1003	Zimmer, Fred	1957/03/12	Sales rep	45000

The database is a collection of ROWs or wcoRDs (lines), each divided into **ATTRIBUTES** (FIELDS). Each attribute has a name. One specific field—in this case, the ID number—is the **KEv** or iDENTIFIER; it is unique to each record and can be used to identify the record. Formally, each record is a TABLE (an ordered set of a specific number of items), a table is a set of tuples, and each table expresses a RELATioN between the values in its tuples. A true relational database can consist of many tables; a **FLAT-FILE** DATABASE, which is less powerful, consists of one table only. For more on database theory, see Date (1990).

Queries. The main function of a database is to answer QcERiEs (requests for information). Database manipulation has a semantics all its own; regardless of the language used, the meanings of most queries will have the form

Do action A to all records that pass test T.

For example, if the query is

Show me all programmers with salaries over 25000.

then the action is *display* and the tests are *Title — programmer, Salary > 25000*.

Most large databases use STRUCTURED QUERY LANGUAGE (SQL) (pronounced “se-quel”). In SQL the query just mentioned would be expressed as:

```
SELECT NAME, SALARY FROM TABLE1
WHERE SALARY > 25000

AND TITLE = 'PROGRAMMER'
```

In this chapter, however, we will use a database consisting of Prolog facts and construct

2.3.3 A Database in Prolog

In order to experiment with natural-language access to databases, we need a database. We will construct a simple database in Prolog, but it will be designed to act like a single table rather than to use Prolog in the most effective way. In this respect the Prolog system will act as if it were accessing a large database implemented in some other language.

First, the records themselves:

```
employee(1001,'Doe, John P.', [1947,01,30], 'President', 100000).
employee(1002,'Smith, Mary J.', [1960,09,05], 'Programmer', 30000).
employee(1003,'Zimmer, Fred', [1957,3,12], 'Sales rep', 45000).
```

Next, predicates to retrieve the values of the individual attributes, given the key:

```
full_name(ID,N) :- employee(ID,N,_,_,_).
birth_date(ID,B) :- employee(ID,_,B,_,_).
title(ID,T) :- employee(ID,_,_,T,_).
salary(ID,S) :- employee(ID,_,_,_,S).
```

Finally, predicates to do other things to the records—in this case, merely display them and remove them:

```
display_record(ID)
    employee(ID,N,B,T,S),
    write([ID,N,B,T,S]),
    nl.
```

```
remove_record(ID) :-  
    retract(employee(ID,_,_,_)).
```

A fully functional database would also have a way to **UPDATE** (change) the attributes in each record.

Exercise 2.3.3.1

Make up a tiny relational (not flat-file) database and implement it in Prolog clauses. Include at least two tables and give an example of a query that does something useful by accessing both tables. Indicate which attribute is the key in each table.

Exercise 2.3.3.2

What is the relationship between first—argument indexing (see Appendix A) and the concept of a key in a database?

2.3.4 Building a Keyword System

Keyword systems work well for database querying because each important concept associated with the database has a distinctive name. A keyword system responds only to the words that identify fields, values, comparisons, and the like, and ignores all the other words.

Like a template system, the keyword system can have *simplify* and *translate* stages. However, the *simplify* stage is relatively unimportant and is often unnecessary. Its only function is to make synonymous words look alike. There is no need for *simplify* to drop unnecessary words such as *the*, because *translate* will drop all the words it does not recognize.

The *translate* stage does most of the work. The output of *translate* will be a Prolog query that picks out the appropriate records and performs some action on them. We want to act on all the records that pass the test, so we will attempt as many alternative solutions to the query as possible.

The main procedure will therefore look something like this:

```
% process_queries

% Accepts database queries from the keyboard and
% executes all solutions to each query.

process_queries :-
    repeat,

    write('Query: '),
    read atomics(Words),

    % some systems would have a 'simplify' stage here,
    translate(Words,_,Query),
```



```

write(Query), nl,                                % for testing
do all_solutions(Query),
Words == [quit],

% do all_solutions(+Query)
% Makes execution backtrack through all solutions to Query.

do all_solutions(Query) -
    call(Query),

    fail.

do_all_solutions(_).

```

The anonymous variable in the arguments of `trans l a t e` will be explained later.

Exercise 2.3.4.1

2.3.5 Constructing a query

From the keyword system's point of view, there are only five kinds of words:

- ACTIONS such as *display*, *erase*, *select*, etc. (we may want to supply *display* as the default action for queries that do not name an action);
- TESTS Such as *programmer*, *president*, *male*, *female*, etc., each of which requires a specific value in a specific attribute;
- COMPARISONS such as *greater*, *over*, *under*, etc., each of which takes two arguments;
 - o **ARGUMENTS** Of the comparisons, such as *salary* (which retrieves the value of a field) and *25000* (which is a constant); and
 - o NOISE WORDS, i.e., unrecognized words which can be skipped.

The string of words must be translated into a Prolog query. This is done by working through it, word by word, and adding something to the query for each meaningful word. Assuming the input list is [*show*, *programmer*, *salary*, *over*, *25000*], the process should go roughly as follows:

Word	Query
<i>show</i>	<code>display*record(X)</code>
<i>programmer</i>	<code>title(X,programmer), display*record(X)</code>
<i>salary</i>	<code>salary(X,Y),</code>
<i>over</i>	<code>Z=25000, Y>Z, display*record(X)</code>

Notice that the action goes at the *end* of the query because we don't want it to be executed until all the tests have been fulfilled. All the other tests, however, come at the beginning, roughly in the order in which the human user gave them. Further, the comparison *salary over 25000* translates into three queries (or in any case two), not just one.

Moreover, the variable x , identifying the record to be retrieved, is shared by all the goals in the query. Otherwise there would be nothing to make them all apply to the same record.

We will put the query together by joining the goals with commas. Recall that if x , y , and z are Prolog queries, then so is (x, y, z) . The comma is an infix operator and is right associative so that $(x, y, z) = (x, (y, z))$. Prolog executes a query of the form (a, b) by executing a and then b .

Knowing this, we can write rules to translate words into queries. Specifically, we can define a predicate with a clause for each word that will translate that word and then call the predicate recursively to translate the rest of the list, thus:

```

translate([showWords],X,(Queries,display_record(X)))      -

        translate(Words,X,Queries) .

translate([programmerWords],X,(title(X,programmer),Queries))

        translate(Words,X,Queries) .

```

The first of these puts an action at the end of the compound query; the second puts an ordinary test at the beginning. Together these two rules translate

```
[show,programmer,...]
```

into

```
((title(X,programmer),...),display_record(X)
```

where ... stands for the rest of the list and its translation, respectively.

Any word for which there is no specific translation should be skipped. Together with the cuts in the previous clauses, the following clause takes care of this:

```
translate([ lWords],X,Query) :-translate(Words,X,Query) .
```

Finally, what happens at the end of the list? It would be handy if Prolog had a “null” or “empty” query that could serve as the translation of an empty list. Sure enough, there is such a thing—the built-in predicate `! true`, which does nothing, but always succeeds. Thus we can write

```
translate([],_,true).
```

and then `trans take` will render `[ show, programmer ]` (with no subsequent words) as:

```
((title(X,programmer),true),display_record(X))
```

It takes less time to execute the redundant `! true` in the query than to perform extra checks and avoid putting it there.

Exercise 2.3.5.1

Get `! trans take` working in the form shown. Add at least three more words to its vocabulary and demonstrate that it works correctly.

Exercise 2.3.5.2

Modify rank fat e so that it prints a warning when a word more than five letters long is ignored. (A more sophisticated system might have a list of hundreds of words that it can safely ignore, and print a warning whenever it ignores a word that is not on this list.)

2.3.6 Lambda Abstraction

The alert reader will have noticed two things. First, we haven't tackled comparisons and their arguments yet. Second, the above clauses for `L` and `ans` contain a mysterious variable `X` whose only purpose is to get unified with all the variables in the query, so that they all become the same variable. As you might guess, the task of keeping the variables straight gets complicated when we start dealing with arguments of comparisons.

Accordingly, we're going to pull out one of our most powerful tools: LAMBDA

ABSTRACTION.

Lambda abstraction (or LAMBDA CALCULUS) may be the most useful technical tool in all of semantics. It was introduced by the logician Alonzo Church (1941) as a way to turn formulas into properties.

In formal logic, *mortal Socrates*) means *Socrates is mortal*, and *mortal Plato*) means *Plato is mortal*. So how do you say simply *mortal*? According to Church, the property of being mortal is expressed by the formula

$$\lambda x.(mortal\ x)$$

where λ is the Greek letter lambda, and (λx) means that x in the formula is to be supplied as an argument.

Here's a more familiar example. In ordinary mathematics we "define" a function f by writing a formula such as

$$\text{Let } f(x) = 4x + 2.$$

But this is really a definition, not of f , but of $f(x)$; the reader is left to assume that the definitions of $f(y)$, $f(z)$, and $f(a + b + c)$ are analogous, with y or z or $a + b + c$ substituted for x . If, instead, we say

$$\text{Let } f \text{ --- } \lambda x.(4x + 2).$$

lambda abstraction makes it explicit that z is not part of the definition, but merely a stand-in for a value to be supplied from elsewhere.

The programming language Lisp uses lambda expressions regularly; in fact it is built around them. A function definition such as

```
(DEFUN F(X) (+ (* 4 X) 2))
```

was written, in the earliest dialect of Lisp, as:

```
IDEFINE F (LAMBDA (X) (+ (* 4 X) 2))
```

That is, the definition of F is a lambda expression that says, “Give me a value—I’ll call it X —and I’ll multiply it by 4, add 2, and give you the result.” Then F is the name of this function, and the lambda expression describes the function.

Exercise 2.3.6.1

Assuming that *studies(plato, philosophy)* means ‘Plato studies philosophy’, use lambda abstraction to write expressions that mean:

1. ‘studies philosophy’
2. ‘Plato studies’
3. ‘studies’

Exercise 2.3.6.2

What is the result of supplying the argument 200 to the expression $(Xq)q' - q - 35$? Distinguish supplying the argument from evaluating the expression.

2.3.7 Lambdas in Prolog

Now back to the main story. Because Prolog is built around predicates rather than functions, the need for lambda expressions does not arise immediately, and there is no standard notation for them. Nonetheless, they are easy to construct and use. While we’re at it, we’re going to extend lambda abstraction beyond its original use, and use lambdas to indicate any part of a formula that is supplied from outside, whether or not our intent is to define a function or a property.

To see the need, recall that we defined *programmer* as $\text{Lit le } (X, \text{programmer})$, and *show* as $\text{display\_record } (X)$. In order to translate a query such as *show all programmers*, we need to put the definitions of *show* and *programmer* together in such a way that the X ’s in them are the same variable. But this isn’t easy. The structures $\text{Lit le } (X, \text{programmer})$ and $\text{display\_record } (X)$

have different principal functors; they don't unify with each other, and there is no easy way to get the X's in them unified together.

At this point we have two options. We could use `= ..'` to decompose terms into lists and then search for the variables in the lists. Or—more simply—we could keep another copy of the variable in an easily accessible place outside the term.

That's where lambda expressions come in. A lambda expression is merely a two-argument term whose arguments are a variable and a term in which that variable can occur. For example:

```
lambda(X, title(X, programmer))  
lambda(X, display_record(X))
```

Notice that lambda doesn't mean anything in Prolog; it's just an arbitrary functor that I picked. It's also rather long. To be more concise, let's use the character `λ` as an infix

3.1 PHRASE STRUCTURE

3.1.1 Trees and PS Rules

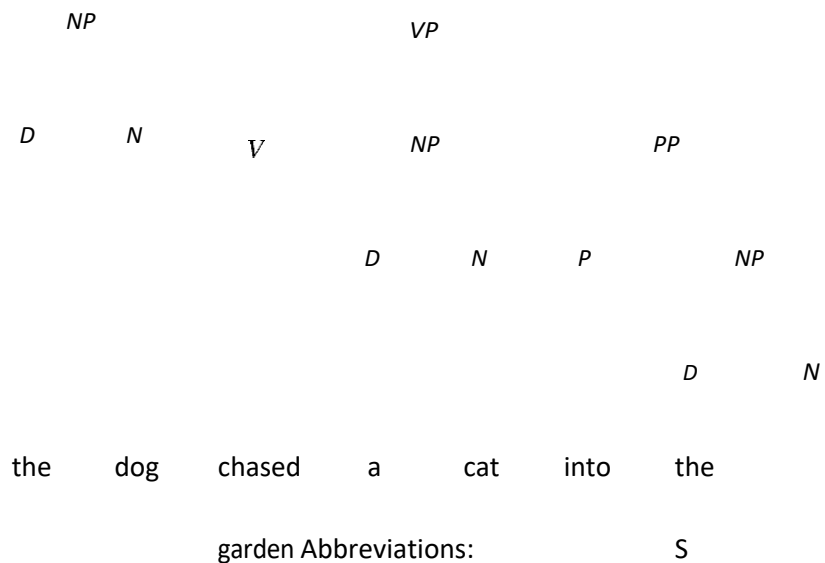
Templates and keywords ignore the role of CONSTITUENT STRUCTURE in human language. That is, they fail to recognize that a sentence is not just a string of words—the words are grouped into phrases, each of which consists of shorter phrases. This is a serious loss, because many of the important properties of language are organized around constituent structure.

Look for example at Figure 3.1. This tree diagram does two things:

- It groups the words into cONSTITUENTs such as *the dog* and *into the garden*.
- It gives names to the constituents, such as “noun phrase” and “prepositional phrase.”

For example, the diagram claims that *into the garden* is a constituent and *a cat into* is not a constituent. This claim appears to be true. You can omit *into the garden*, or replace it with a single word such as *anywhere*, without grossly changing the structure of the rest of the sentence. You can't do this with *a cat into*.

The diagram also claims that *the dog*, *a cat*, and *the garden* are phrases of the same kind (they are all labeled NP). This, too, appears to be a reasonable claim; these



garden Abbreviations:

Sentence

- NP Noun Phrase
- VP Verb Phrase
- PP Prepositional
 Phrase D
- Determiner
- N Noun
- V Verb
- P Preposition

Figure 3.1 Constituent structure of a simple sentence.

three phrases have many syntactic and semantic properties in common. Indeed, you get grammatical sentences if you interchange them with each other.

Contrast all of this with a template or keyword system. Neither templates nor keywords break the sentence into constituents in any systematic way, and neither of them has a clear way to say that two sequences of words, found in different places, are constituents of the same type.

The structure shown in Figure 3.1 is sanctioned, or GENERATED, by the following set of PHRASE-STRUCTURE RULES (PS RULES):

<i>S</i>	--r	<i>NP VP</i>
<i>NP</i>	→	<i>D N</i>
<i>VP</i>	--r	<i>V NP</i>
<i>VP</i>	--r	<i>V NP PP</i>
<i>PP</i>	--r	<i>P NP</i>
<i>D</i>	→	the
<i>D</i>	→	a
<i>N</i>	→	dog

V saw
P into

The rules mean: “An S can consist of an NP followed by a VP. An NP can consist of a D followed by an N. A VP can consist of a V followed by an NP” (and so on).

A set of rules such as these is called a GRAMMAR. We say that the grammar GENemTEs every sentence whose structure is entirely consistent with the rules. For example, this grammar also generates the sentence *The cat saw the dog* assigning it the structure shown in Figure 3.2. A grammar that generates sentences by means of explicit rules, rather than merely describing them in some other way, is called a GENERATIVE GRAMMAR. Ultimately, we would like to discover all the phrase—structure rules of English (together with any other kinds of rules that might be needed) and build a complete generative grammar of English.

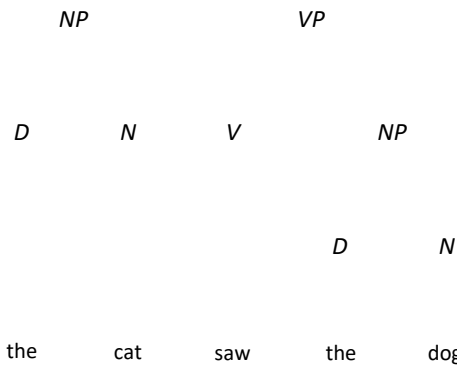


Figure 3.2 The tree structure of a sentence
Shows the rules by which it is generated.

Exercise 3.1.1.1

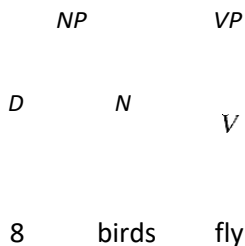
Give tree diagrams of three more sentences that are generated by the grammar just given.

3.1.2 Phrase-Structure Formalism

Formally, each phrase—structure rule has a NonTERMINAL SYMBOL On the left (such as *S*, *NP*, etc.) and an Expansion Of this symbol on the right. The expansion consists of a series of symbols which may be either terminal or nonterminal. The TERMINAL SYMBOLS are the words that actually occur in the language, such as *cut* and *saw*.

Commonly, there is more than one rule expanding the same symbol. This means that more than one expansion is permissible. For example, in the grammar just given, *N* can be either *car* or *dog*.

A symbol can even expand to nothing at all. According to one popular analysis, the structure of *Birds fly* is:



According to this analysis, the sentence begins with a determiner but you can't see it or hear it; it is the NULL (silent) determiner, introduced by the rule

$$D \rightarrow \emptyset$$

Not all linguists accept this analysis, but the formalism provides for it.

Parentheses in a phrase—structure rule denote items that can be left out. For example, the rule

$$VP \rightarrow V (NP)$$

is really an abbreviation for two alternative rules:

$$\begin{array}{l} VP \rightarrow V NP \\ \quad \rightarrow \end{array}$$

That is, the rule says that a VP consists of a V which may or may not be followed by an NP.

All the phrase-structure rules that we have discussed are coTEXT-FREE because each of them can apply in any context. For example, the rule *NP* *D N* asserts that any NP can consist of D and N, regardless of where it occurs. The alternative would be coTEXT-SENSITIVE rules such as “NP expands to D and N, but only when preceded by a preposition.” Context-sensitive rules are rarely used in natural-language analysis; they are potentially too complicated and do not lend themselves to a straightforward parsing algorithm.

Some older textbooks refer to generation of a sentence as if it were a process:

“Start with an S. Use the first rule to rewrite S as NP VP. Use the second rule to rewrite NP as D N,” and so on. This is a misleading way to use the rules because these are certainly not the steps that the human brain goes through when constructing a sentence. (You don’t start out by deciding to utter an S, then realize you need an NP and a VP, and then choose your words only at the very last step.) The purpose of the rules is to specify what structures are possible. The rules themselves are not a procedure; rather, many different procedures for creating or analyzing sentences can be built upon them.

Exercise 3.1.2.1

Draw a tree for *The cat saw a problem* as generated by the following grammar:

<i>S</i>	--r	<i>NP Aux VP</i>
<i>Aux</i>	→	<i>B</i>
<i>NP</i>		<i>NgrouP</i>
<i>NgrouP</i>	--r	<i>D N</i>
<i>VP</i>	→	<i>V NP</i>
<i>D</i>	---+	<i>a</i>
<i>D</i>	→	<i>the</i>
<i>N</i>	--r	<i>problem</i>
<i>N</i>	--r	<i>cat</i>
<i>V</i>	→	<i>puzzled</i>
<i>V</i>	---+	<i>saw</i>

Exercise 3.1.2.3

The grammar in the previous exercise is needlessly complex. Simplify it. After simplification, it should still generate exactly the same sentences and group the words into constituents in the same way. The tree diagrams need not be the same in every detail (in fact if you change the grammar at all, they won't be). Give the tree for *The cat saw a problem* as generated by your simplified grammar.

3.1.3 Recursion

Besides being able to describe constituency, phrase—structure rules have another big advantage over templates and keywords: they can be recursive.

Recall that a template system has to match the entire sentence to a pattern. Consider now the four sentences:

The dog chased the cat.

The girl thought the dog chased the cat.

The butler said the girl thought the dog chased the cat.

The gardener claimed the butler said the girl thought the dog chased the cat.

Obviously, there's a recursive pattern here: the longer sentences have shorter sentences within them. Specifically, a verb such as *thought*, *said*, or *claimed* can introduce another complete sentence.

Templates can't describe a recursive structure like this.¹ The best we could do, with templates, would be to have a separate template for each of the sentence structures above. But that's not satisfactory, for two reasons:

- It simply ignores the important fact that all four sentences are examples of the same repetitive pattern.

'Bobrow's STUDENT could; it was a template system with recursion added. But this is not the way templates are normally used.

- There's no limit to the number of templates that might be needed, because the process can be continued almost without limit, producing longer and longer sentences.

The second of these is the more important. A sentence of this kind could have five verbs, or ten, or maybe even fifteen. It won't have 10,000, but that's because of limits on people's patience and memory capacity, not because of any rule of grammar. This is what linguists mean when they say that a language has a potentially infinite number of different sentence structures.

PS rules solve the problem nicely. Recall that so far, the grammar we are working with is:

<i>S</i>		<i>NP VP</i>
<i>NP</i>	--r	<i>D N</i>
<i>VP</i>		<i>V NP (PP)</i>
<i>D</i>	--r	the
<i>D</i>		a
<i>N</i>		dog
<i>N</i>		cat
<i>N</i>		garden
<i>Y</i>	→	chased
<i>V</i>	--r	saw
<i>P</i>		into

To get recursive structures, all we need to do is add a few more rules:

	→	
<i>N</i>	--r	gardener
<i>N</i>	→	girl
<i>N</i>		butler
<i>V</i>	→	claimed

V	→	thought
V		said

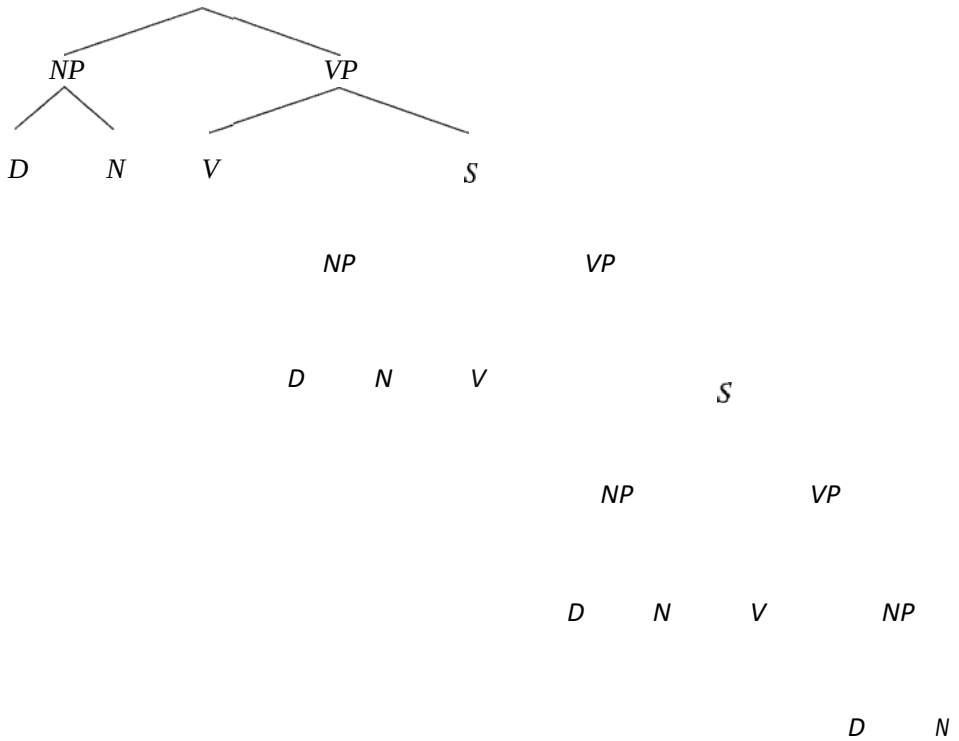
The crucial change here is that now a VP can contain an S. Because every S contains a VP, this allows there to be sentences within sentences. Now the grammar can generate sentences such as *The butler said the girl thought the dog chased the cat* (Figure 3.3).

Exercise 3.1.3.1

How long is the longest sentence that can be generated by the grammar just given?

Exercise 3.1.3.2

Give two sentences that are not correct English but are generated by the grammar as given so far. Draw trees for these sentences.



the butler said the girl thought the dog chased the cat

Figure 3J Recursive PS rules can generate sentences within sentences.

Exercise 3.1.3.3

Linguists consider it very important that a generative grammar must not generate ungrammatical sentences (sentences that cannot occur in the language). If it did, it could not be a correct description of the language, nor of any subset of it.

Is this requirement equally important in natural language processing? Explain.

3.2 TOP-DOWN PARSING

3.2.1 A Parsing Algorithm

To PARSE a sentence is to determine, by an algorithmic process, whether the sentence is generated by a particular grammar, and if so, what structure the grammar assigns to it. (The first few parsers that we build will not give the structure; do not be alarmed. They *find* the structure even though they do not report it.)

There are two major ways to parse:

- o A BOTTOM-its parser looks at the words and tries to combine them into constituents, like this:

n The is a D.

- *Dog* is an N.
- D and N together make an NP . . .

- A TOP—DOE parser starts out looking for a specific constituent, such as *S*, and uses the rules to figure out what this can consist of:
 - *S* consists of NP and VP.
 - NP consists of D and N.
 - *The* is a D.
 - *n Dog* is an N

Both top—down and bottom—up parsers need the ability to BACKTRAcK, i.e., try one rule, and if it doesn't work out, back up and try a different one. A parser that backtracks is called a NONDETERMINISTic parser.

In this chapter we will concentrate on nondeterministic top-down parsing. One way to parse top—down is to give each of the PS rules a procedural interpretation. For example,

- *S --r NP VP* means "To parse an *S*, parse an NP and then a VP."
- *N --r dog* means "To parse an N, accept the word *dog* from the input string."

(The input to a parser is always called the INPUT STRING even though in Prolog it is usually a list of atoms, not a string of characters.) This process is often called RECURSIVE DESCENT because each PS rule is made into a procedure, and these procedures can call each other or themselves recursively.

Parsing then becomes a process very much like satisfying Prolog queries. The initial goal is *S*; some of the rules transform this goal into other goals; and some of the rules satisfy goals by accepting words. Figure 3.4 shows the complete process of parsing *The dog saw the cat*.

In each step the parser either transforms the current goal into a new goal, or satisfies the current goal by accepting a word. Sometimes, although there is no example of it here, the parser backtracks. In all these respects top-down parsing is very much like the way Prolog solves queries. This is not a coincidence; parsing

is one of the applications that Alain Colmerauer had in mind when he originally invented Prolog (Colmerauer 1978).

3.2.2 Parsing with Prolog Rules

To implement a parser, all we have to do is rewrite the grammar as a set of Prolog clauses. A grammar written in this form is called a DEFINITE—CLAUSE GRAMMAR. “Definite clauses” are merely Prolog rules and facts; definite—clause grammars were introduced to the world by Pereira and Warren (1980), though they were implicit in earlier logic programming research.

In a definite—clause grammar, each PS rule is a clause for a predicate with two

arguments, like this:

S *NP VP*

s (*L1*, *L*) :- *np* (*L1*, *L2*) , *vp* (*L2*, *L*) .

Start with:	S	<i>the dog saw the cat</i>
Apply S → NP VP	NP VP	<i>the dog saw the cat</i>
Apply NP → D N	D N VP	<i>the dog saw the cat</i>
Apply D → the	the N VP	<i>the dog saw the cat</i>
Accept the	N VP	<i>dog saw the cat</i>
Apply N → dog	dog VP	<i>dog saw the cat</i>
Accept dog	VP	<i>saw the cat</i>
Apply VP → V NP	V NP	<i>saw the cat</i>
Apply V → saw	saw NP	<i>saw the cat</i>
Accept saw	NP	<i>the cat</i>
Apply NP → D N	D N	<i>the cat</i>
Apply D → the	the N	<i>the cat</i>
Accept the	N	<i>cat</i>
Apply N → cat	cat	<i>cat</i>
Accept cat	(empty)	(empty)

Figure 3.4 Top—down parsing. Each action changes either the input string or the goal list, or both.

Here:

- L1 is the original input string, such as [the , dog , s aw, the , cat] ;
- L2 is the input string without the initial NP, such as [s aw, the , cat] ; and
- L is the input string without the NP or the VP (in this case, []).'

The rule itself could be expressed in English as:

To parse an S starting with input string L1 and ending with L, first parse an NP starting with L1 and ending up with L2, then parse a VP starting with L2 and ending up with L.

Or it could be expressed more declaratively as:

Removing an S from L1 gives

L if

removing an NP from L1 gives L2

and

removing a VP from L2 gives L.

The rules that introduce terminal symbols (words) take the form

$n([dog1L], L) .$

'Alternatively, L1, L2, and L could denote *positions* in the input string: the beginning of the string, the end of the NP, and the end of the VP. But the aforementioned interpretation is more common.

That is, “To parse a noun, remove *dog* from the beginning of the input string.” So a complete grammar, in Prolog, looks like this:

```
s(L1,L)   :- np(L1,L2), vp(L2,L) .
np(L1,L)  :- d(L1,L2),  n(L2,L) .
vp(L1,L)  :- v(L1,L2),  np(L2,L) .
d([the1L],L) .
d([a1L],L) .
n([dog1L],L) .
n([cat1L],L) .
n([gardener1L],L) .
n1[policeman1L],L) .
n([butler1L],L) .
v([chased1L],L) .
v([saw1L],L) .
```

All of this deserves careful study. Take a moment to convince yourself that a query such

as

```
?- s((the,dog,chased,the,cat), []).
```

really will succeed if and only if, according to the available Rules, *the dog chased the cor* is a sentence. Prolog automatically chooses the right rules and backtracks where necessary.

Exercise 3.2.2.1

Get this parser working on the computer and use it to parse several sentences.

Exercise 3.2.2.2

Is this kind of parsing reversible? That is, what happens if the input string is uninstantiated? Try it and see.

Exercise 3.2.2.3

Show how to use this parser to “fill in the blanks” in a template such as *MW, X, Y, Z, dog*). That is, give this template to the parser and make the parser generate a five-word sentence that ends with *dog*.

3.3 DCG RULES

3.3.1 DCG Notation

The conversion of PS rules into Prolog clauses is so simple that it can be done automatically. In fact Prolog includes a facility for doing just this. If you write clauses in

```
np --> d, n.
n --> [dog].
```

then, upon reading the clauses, consult and reconsult so it will convert them automatically into

```
s(L1,L) :- np(L1,L2), vp(L2,L).
np(L1,L) :- d(L1,L2), n(L2,L).
n([dog|L],L).
```

or the equivalent.' This conversion is done when the clauses are loaded from a file into memory. Once they are in memory, the Prolog system completely forgets that they were ever expressed in DCG notation. That's why, if you type ?-listing.' or

?-listing np/2.' you will see the translated rules, not the rules you wrote.

The DCG translator affects only the clauses with the principal functor -->. All other clauses are assumed to be plain Prolog and are left unchanged. Thus your program can contain both DCG rules and plain Prolog clauses.

Here is an example of a grammar in DCG notation:

```
s --> np, vp.
np --> d, n.
vp --> v, np.
d --> [the];[a].
n --> [dog];[cat];[gardener];(policeman);[butler].
v --> [chased]:[saw].
```

Every DCG rule takes the form

nonterminal symbol --> *expansion*

where *expansion* is any of the following:

- A nonterminal symbol such as `np`;
- A list of terminal symbols such as `[ dog ]` or `[ e act, ofher ]`;
- A null constituent represented by `[ ]` ;
- A plain Prolog goal enclosed in braces, such as `(or i t e ( ' Found NP ) )` ;
- A series of any of these expansions joined by commas.

If your Prolog requires all the clauses of a predicate to be contiguous, then the same requirement will apply to DCG rules; for example, you can't have an `np` rule followed by a `vp` rule and then another `up` rule.

[†]These translations are used in Arity Prolog and several of the older Prolog implementations. The translations used by Quintus are quite different and are explained in Section 3.3.3.

As in Prolog, the semicolon means 'or.' This is convenient in writing rules such as

```
n --> [dog];[cat];[butler].
```

Apart from this, we avoid using the semicolon because it is potentially confusing. (The vertical bar, '|', is equivalent to the semicolon in Quintus Prolog.)

The translator turns DCG rules into Prolog clauses by adding two extra arguments to every symbol that is not in brackets or braces. These arguments are automatically arranged appropriately for parsing. That's how `np` in the DCG rule becomes something like `np t L1, L2` in the resulting Prolog clause.

Notice that the automatically generated arguments are *not* the same as any variables supplied by the programmer. (They are not named `L1`, `L2`, and `L`; in fact they have no names at all.) A common mistake is to write something like

```
s --> np, ( write(L2) ), vp.      % WRONG!
```

as if `L2` meant something here. It doesn't.

Exercise 3.3.1.1

Write `D → 7` as a DCG rule,

Exercise 3.3.1.2

What is your Prolog system's translation of each of the following PS rules? (See Section

S.3.3 for an explanation of why some of them may not be what you expect.)

```
s --> np, vp.
```

```
n --> [dog].
```

Exercise 3.3.1.3

Get the grammar on p. 46 working and use it to parse a few sentences.

Exercise 3.3.1.4

What is the difference between the following two rules?

```
n --> [dog] .  
n --> dog .
```

Exercise 3.3.1.5

Here is a devious way of getting access to the input string during parsing:

```
snoc'p { X, Y, X, Y } .  
s --> np, snoop (A, B), ( write (A), A=B ), vp .
```


only two arguments in a DCG clause, because the DCG translator adds arguments. The rest of the grammar is above.

Explain what snoop does and why it works. Show how snoop could be used to modify the input string during parsing.

3.3.2 Loops

Like all left-to-right top-down parsers, DCG—rule parsers go into a loop when they encounter a rule of the form

$$A \rightarrow A B$$

—that is, “To parse an A, parse an A and then . . .”

The problem, of course, is that this rule calls itself before it has accepted anything from the input string, so there is nothing to keep it from calling itself again and again, ad infinitum. A loop of this kind could be spread over two or more rules, such as:

$$\begin{array}{lcl} A & \text{--r} & B \\ B & \text{--+} & A C \end{array}$$

Rules or sets of rules like these are called LEFT-RECURSIVE.

Left-recursion occurs whenever a constituent begins with another constituent of exactly the same kind. Figure 3.5 shows some examples in English. The left—recursive rule

$$NP \quad NP \text{ Conj } NP$$

and the left-recursive rule pair

$$\begin{array}{lcl} NP & \rightarrow & D N \\ D & & NP \text{ Poss} \end{array}$$

appear to be necessary to describe English sentences correctly.

It has long been known that any left—recursive grammar can be transformed into another grammar that generates the same strings but is not left-recursive. For example,

$$\begin{array}{ll} NP & \rightarrow \quad NP \text{ Conj } NP \\ NP & \quad \quad D \ N \end{array}$$

can be changed to:

$$\begin{array}{ll} NP & \rightarrow \quad NPX \text{ Conj } NP \\ NP & \quad \quad NPX \\ NPX & \quad \quad D \ N \end{array}$$

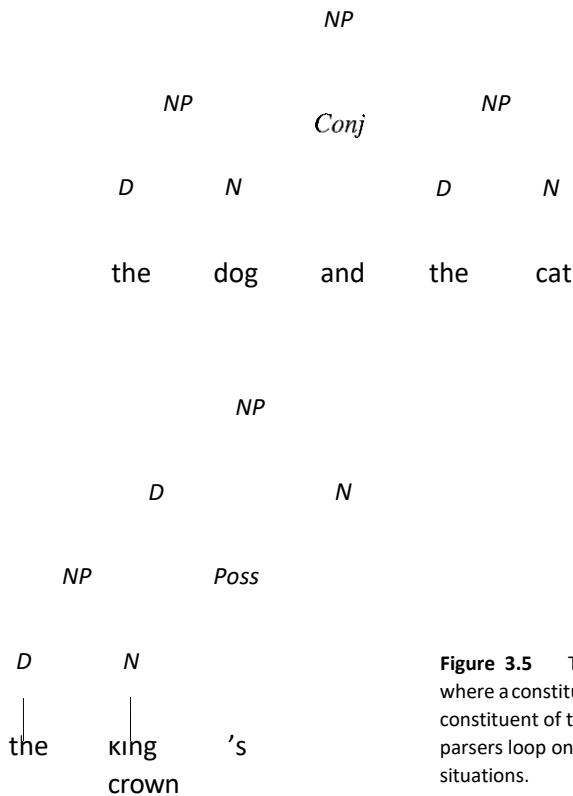


Figure 3.5 Two structures in English where a constituent begins with a constituent of the same type. Top-down parsers loop on the rules required for such situations.

Unfortunately, although this generates the right strings, it gives a tree structure that is not entirely right. There is no linguistic evidence that the distinction between NP and NPX actually exists in the language.

Transforming the grammar, then, is not too useful in natural language processing. We prefer to switch to a different parsing algorithm (Chapter 6) or make minor patches in our top-down parsing algorithm to avoid loops while keeping the tree structure correct.

Exercise 3.3.2.1

What is the tree structure of the sentence *dog and the cat* according to a grammar containing the rules with NPX just discussed?

Exercise 3.3.2.2

Implement the rules

$$\begin{array}{lll} NP & \rightarrow & NP \text{ Conj } NP \\ NP & \rightarrow & D \ N \end{array}$$

in a DCG parser in such a way that no loop occurs. To do this, use `snoop` from the previous section, so that your implementation of $NP \rightarrow NP \text{ Conj } NP$ will do the following:

This is not a theoretically satisfying way to process conjoined noun phrases, but it works.

3.3.3 Some Details of Implementation

Two details of implementation remain to be considered. First, in most Prologs the DCG translator is actually a built—in predicate named `expandpterm`.⁴ Before asserting clauses into memory, `consulL` and `econsulL` pass every clause to `expandpterm`. If `expandpterm` succeeds, its output is used in place of the clause that was passed to it. Otherwise the original clause is used.

This is important because you can invent your own translator (for DCG rules or any other kind of clauses) and implement it by adding clauses to `expandpterm`. See your Prolog manual for details.

Second, Quintus Prolog has a special way of translating DCG rules that contain terminal symbols (words). Instead of rendering

```
n --> [dog].
```

as

```
n([dog|L],L).
```

Quintus renders it as

```
n(L1,L) :- 'C'(L1,dog,L).
```

where 'C' is defined as

```
'C'([X|Y],X,Y).
```

Here 'C' stands for “connects” and was in fact called `connects` in early Edinburgh implementations.

At first sight the use of 'C' may seem to be just a waste of time. Why not match the lists directly instead of calling another predicate? The answer, as O’Keefe (1985) explains, is that 'C' makes a difference when there is an embedded Prolog goal with side effects. Consider for instance the rule

```
p --> [because], ( write('Got it!') ), [of].
```

In ALS Prolog, `expand` term is called `ins:dcgexpand` and you have to consult `(dcgs)` before using it.

This means “To parse a preposition, accept *because*, then write ‘Got it!’, then accept *of*.” The idea is that the two words *because of* should be treated as a single preposition, and furthermore, perhaps for debugging purposes, we want to see a message when the parser accepts *because*.

In Quintus Prolog, this rule translates to

```
p(L1,L) :- 'C'(L1,because,L2), write('Got it!'), 'C'(L2,of,L).
```

and works as intended. However, in Prologs that use the old-style translation, such as Arity Prolog 4.0, the same rule translates to

```
p([because,of!L],L) :- write('Got it!').
```

which does not give the intended result—it does not write “Got it!” until both *because* and *of* have been accepted. If, instead of `wri C e ( Not i t ! ' )`, the embedded goal had been a cut, the flow of execution could have ended up substantially different.

ALS Prolog solves the same problem a different way, by translating the rule as:

```
p([because!L1],L) :- write('Got it!'), L1 = [of!L].
```

The ALS translator knows which unifications are supposed to be done after executing the embedded goal, and postpones them appropriately.

Exercise 3.3.3.1

What are the results of the queries

```
?- expand_term((s → np, vp), What).  
°- expand_term(green(kermit), What).
```

in your Prolog? (If neither one of them succeeds, find out how to use `expand_L_erm` or its equivalent in the implementation you are using.)

Exercise 3.3.3.2

How does your Prolog translate the following rule?

```
p --> (write('Processing because-of')), [because,of].
```

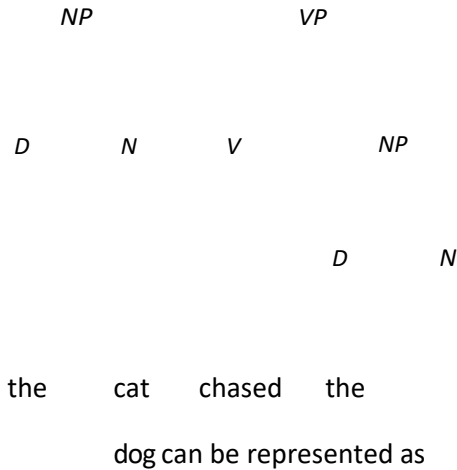
Explain how you found out. What are the advantages and disadvantages of using this translation?

3.4 USING DCG PARSERS

3.4.1 Building Syntactic Trees

A practical parser should do more than just say whether or not a sentence is acceptable; it should also report the structure of the sentence. This raises two questions: how to

example, the tree



the structure:

`s(np(d(the), n(cat)), vp(v(chased), np(d(the), n(dog))))`

To produce this representation, the parser will make each rule fill in the part of the structure for which it is responsible. For example, parsing begins with the rule $s \rightarrow np, vp$. This rule must therefore contribute the outermost $s(\dots, \dots)$ in the structure, where the portions represented by $\dots$ will be filled in by the np and vp rules, respectively. The np rule in turn will contribute $np(\dots, \dots)$, with arguments to be supplied by rules yet lower down.

To implement this process, we'll rely on the fact that DCG notation allows extra arguments on predicates. If, for example, you write

`s(a,b) --> np(c,d), vp(e,f).`

the translator will produce:

```
s(a,b,L1,L) :- np(c,d,L1,L2), vp(e,f,L2,L).
```

These extra arguments make a DCG more powerful than an ordinary phrase-structure grammar. (Normally, the extra arguments that you supply are placed before the arguments supplied by the translator, but it's wise to check your implementation.)

In the present case we want the arguments to represent the tree, so the syntactic rules need to look like this:

```
s(s(NP,VP)) --> np(NP), vp(VP).
```

```
np(np(D,N)) --> d(D), n(N).
```

```
vp(vp(V,NP)) --> v(V), np(NP).
```

```
d(d(the)) --> [the].
```

```
n(n(dog)) --> [dog].
```

```
n(n(cat)) --> [cat].
```

```
v(v(chased)) --> [chased].
```

```
v(v(saw)) --> [saw].
```

When the first rule is invoked, its argument is immediately instantiated as `s (NP , VP )`, but the variables `NP` and `VP` are not yet instantiated. The `np` rule then instantiates `NP` to `np ( D , N )` so that the whole structure is `s (np ( D , N ) , VP )` but `D`, `N`, and `VP` do not yet have values. The structure will be completely instantiated when parsing is complete. Moreover, if execution backtracks out of a rule, the instantiations established by that rule are undone, just as in ordinary Prolog.

The key idea here is that unification and instantiation give you a way to *work with information that you do not yet have*—just build a partly instantiated structure and instantiate the details later. This technique gives Prolog much of its power.

Exercise 3.4.1.1

Get this parser working, then extend it by adding all the rules developed in Section 3.1.3. Show that it generates a correct parse tree for *The butler claimed the dog chased the cat into the garden*.

Exercise 3.4.1.2

Construct a parser that outputs structural information, not by instantiating arguments, but by performing a write at the beginning and end of each rule, like this:

```
s --> { write('Beginning S...') }, np, vp,
      ( write('Finished S.') ) ].
```

This parser will indicate the structure of *The dog chased the erm* by outputting messages like “Starting S . . . Starting NP . . . Determiner . . . Noun . . . End of NP . . . Starting VP . . . ” and so on.

Exercise 3.4.1.3

What is fundamentally wrong with the technique used in the previous exercise? Why is its output not always an accurate description of the tree?

Exercise 3.4.1.4 (project)

Write a Prolog program that reads a file of DCG rules without arguments and automatically converts them into rules that build a syntactic tree, writing the translation on another file. For example,

```
s --> np, vp.
```

should be translated into

```
s(s(V1,V2)) --> np(V1), vp(V2).
```

Write a predicate called `display_Ltree` as outlined that will output a tree structure such as `s (np (d (the) , n (birds) ) , vp (v (fly) ) )` in an indented outline-like format, like this:

```

s
  np
    d
      the
    n
      birds
  vp
    v
      fly

```

thus making it easier to see the tree structure. (This is closely related to the “depth” exercises in Appendix A.)

Exercise 3.4.1.6 (project)

Write a predicate called `display_Ltree` that will output a tree structure such as `s (np (d (the) , n (birds) ) , vp (v (fly) ) )` as an actual tree made of ASCII characters, such as:

```

      np      vp
      / \    / \
     d  n   v
    / \ / \
   the birds fly

```

or the like. This project is not easy, but the result is well worth having.

Another use of arguments is to enforce AGREEMENT. In English, the verb and its subject have to agree in number; that is, they are either both singular or both plural. Most nouns take -s in the plural; most third—person verbs take -s in the singular, like this:

<i>The dog chases the cats.</i>	(Singular subject, singular verb)
<i>The dogs chase the cats.</i>	(Plural subject, plural verb)
<i>*The dog chase the cats.</i>	(Singular subject, plural verb)
<i>*The dogs chases the cats.</i>	(Plural subject, singular verb)

(Here as elsewhere, asterisks indicate ungrammatical examples.) Notice that the object (the noun after the verb) is not involved in agreement; you can say either *cat* or *cats* in any of the sentences above.

To make DCG rules account for agreement, we need to attach arguments to the NP and VP to indicate whether each of them is singular or plural:

```
n(singular) —> [dog];[cat];[mouse].
```

```
n(plural) --> [dogs];[catsJ];[mice].
```

```
v(singular) —> [chases];[sees].
```

```
v(plural) —> [chase] ; (see).
```

The point is that an NP has the same number (singular or plural) as its HEAD NOON, and a VP has the same number as its HEAD VERB. These rules ensure that this is so:

```
np(Number) —> d, n(Number).
```

```
vp(Number) --> v(Number), np(_).
```

The sentence consists of an NP and a VP whose numbers match:

```
s --> np(Number), vp(Number).
```

A parser using these rules will accept only sentences in which the subject and verb agree in number.

Exercise 3.4.2.1

Get a parser working that uses these rules, and verify that it works as intended. Specifically, it should accept *The dogs chase the cats* but not *The dog chase the cats*.

Exercise 3.4.2.2

Why couldn't the VP rule be written as follows?

$$\text{vp}(\text{Number}) \rightarrow \text{v}(\text{Number}), \text{np}.$$

Exercise 3.4.2.3

Extend your parser so that it distinguishes between singular and plural determiners. For example, *o* and *an* are used only with singulars; *two* is used only with plurals; and *the* is used with both singulars and plurals. Your parser should reject **a dogs*, *two dog*, and the like.

Exercise 3.4.2.4

Extend your parser so that it distinguishes between count and mass nouns. Give each noun two features, *Number* (with values *singular* or *plural*) and *C1* as *count*

Some examples:

TYPE OF NOUN	EXAMPLE	SUITABLE DETERMINERS
singular count	<i>dog, theory</i>	a(n), the, one, every
plural count	<i>dogs, theories</i>	B, the, two, all
singular mass	<i>water, stuff</i>	B, the, all
plural mass	(does not exist)	

Your parser should accept a *dog*, *the theory*, *two dogs*, etc., but reject "*a dogs*", *'one theories*, **every stuff*, *'vivo water*, and the like.

3.4.3 Case Marking

Some English pronouns are marked for cAsE. This means that the pronoun has a different form before the verb than after it. For example:

He sees him. **Him sees he.*
She sees her. **Her sees she.*
They see them. **Them see they.*

The forms that come before the verb, *he*, *she*, and *hey*, are called NoMiNATIVE, find the forms that come after the verb, *him*, *her*, and *them* are called AccuSATIVE.'

"Before the verb" and "after the verb" is not a very good way of describing where these forms occur. Instead, we can say, much more precisely and accurately, that:

S --r NP VP introduces a nominative, and

VP V NP introduces an accusative.

To account for this with DCG rules, we'll add a second argument for case, alongside the argument for number. The rules for the pronouns will then be:

```
pronoun(singular,nominative) --> [he]; [she].  
pronoun(singular,accusative)  -> [him]; [her].  
pronoun(plural,nominative)   --> [they].  
pronoun(plural,accusative)   --> [them].
```

and we need to change the NP rules:

```
np(Number,Case) --> pronoun(Number,Case).  
np(Number,_)   --> d, n(Number).
```

'Her is also a possessive determiner corresponding to *his, your, my*, etc. Don't let this confuse you.

The second of these says that nouns are not marked for case. Because there is no case marking on determiners, nouns, or verbs, the D, N, and V rules can be exactly the same as in the previous section. But we need to change the S rule and the VP rule:

```
s --> np(Number,nominative), vp(Number).
vp(Number) --> v(Number), np(_,accusative).
```

Now the parser accounts for case marking of pronouns.

Exercise 3.4.3.1

Incorporate these changes into the rules from the previous section, and verify that it accepts

He sees them and *The y see her* but not *Them see he* or **F/tey see the y*.

Exercise 3.4.3.2

Should the rule $PP \rightarrow P NP$ specify a nominative or an accusative NP? Cite examples from English to justify your conclusion, then add this rule to your parser, with appropriate case marking.

3.4.4 Subcategorization

The structure of the English VP depends on the particular verb. Different verbs require different things after them. For example:

VERB

COMPLEMENT

sleep, bark None
 chase, see One NP
 give, sell Two
 NPs say, claim
 Sentenc
 e

EXAMPLE

(The cat) slept.

(The dog) chased the cat.

[Max] sold Bill his car.

[Max] claimed the cat barked.

If these requirements are not met, the sentence is ungrammatical; you can't say *The dog chased* or *John said the cci* unless the missing part can be clearly understood from the context.

This means we can't really account for the VP with a single rule of the form $VP \rightarrow V (NP) (NP) (S)$. Instead we need at least four rules,

$VP \rightarrow V NP$

$VP \rightarrow V NP NP$

$\rightarrow$

VP	--r	V2 NP
VP		V3 NP NP
VP	--r	V4 S

and the lexicon would say that *bark* is a V1, *chase* is a V2, and so on.

Notice that if we do this, we are claiming that there is no relationship at all between V1, V2, V3, and V4; no more than between N and P. We will be claiming that *bark*, *chase*, *sell*, and *say* are four totally different kinds of words. There will be no way to write a rule that applies equally to all four of them.

That isn't satisfactory. There are two good reasons to put all four kinds of verbs into a single category called V:

- Only V's are marked for tense and number and can serve as the head of a VP.
- Morphologically, all four kinds of V's are alike (they take -s in the third person singular, -ed in the past, and so on). The morphological part of the parser, which we haven't implemented yet, *should* nor distinguish between different kinds of V's.

What we need is a way to wave things both ways: treat the various kinds of V's alike and treat them differently. That is, we need a category V, divided into subCATEGORIES.

One way to do this is—you guessed it—to add an argument to the V. Then rules that care about this feature can specify its value, and rules that don't care about it can put an anonymous variable in place of it. Here goes:

```
vp --> v(1) .
```

```
vp --> v(2), np.
```

```
vp --> v(3), np, np.
```

```
vp --> v(4), s.
```

```
v(1) --> [barked];[slept] .
```

```
v(2) --> [chasedJ;[saw]
```

```
v(3) --> [gave];[sold] .
```

$v(4) \rightarrow [said]; [thought].$

(To keep the example from becoming too complicated, we've dropped the agreement features. A real parser would, of course, keep them.) Now $v(1)$ means a verb of class 1, $v(2)$ means a verb of class 2, etc., and $v(p)$ means a verb of any class. It turns out that $v(p)$ does not occur in any syntactic rules, but only in the morphological part of the grammar.

Exercise 3.4.4.1

Construct a complete, working parser that implements subcategorization of verbs. Make it accept the sentences

The cat slept.

The dog chased the cat.

The girl gave the dog a bone.

The boy said the dog chased the cat.

but reject sentences in which verbs have the wrong kinds of complements, such as:

'The cat slept the dog.

*The dog chased.

*The girl gave the dog chased the cat.

*The boy said the cat.

3.4.5 Undoing Syntactic Movements

You can even use arguments to pick up a word from one position and put it down somewhere else. This surprising ability turns out to be needed when parsing English questions.

Consider a complicated sentence such as:

Max said Bill thought Joe believed Fido barked.

This sentence contains four NPs, *Max*, *Bill*, *Joe*, and *Fido*, and we can ask a question by substituting who for any of them:⁶

*Who said Bill thought Joe believed Fido barked? [Max.] Who
 did Max say thought Joe believed Fido barked? [Bill.]
 Who did Max say Bill thought believed Fido barked? [Joe.]
 Who did Max say Bill thought Joe believed barked? (Fido.)*

The first sentence simply puts who in place of *Max*. In each of the others:

- Exactly one NP is missing from somewhere in the sentence (it is denoted by ‘p’).
- Up to *did* has been added at the beginning.
- The sentence means exactly what it would have meant if who had appeared in place of the missing NP.

We would like the parser to move who back into the NP position with which it is associated. To do this, we will use arguments to implement a HOLDING LII r on which

⁶Like most native speakers, we will ignore the difference between *trio* (old nominative) and *whom* (old accusative). For speakers who still distinguish them, the difference between *it*’/to and *w*/tom is exactly the same

and a rule later on that needs an NP but can't find one will use the stored occurrence of

w/to⁷

To implement a holding list, each rule needs two arguments, one for input and one for output—that is, one to receive the holding list and one to output the (possibly changed) holding list to the next rule.

Only two rules need special treatment here. In addition to the normal rule *S* → *NP VP*, we need a rule to parse sentences that begin with *who*:

```
s(In,Out) --> [who,did], np([who|In],Out1), vp(Out1,Out).
```

That is: *In* is the input received by the whole sentence (probably an empty list). This rule accepts the words *who did*, then passes *i who 1 In* as input to the NP. The output of the NP is one *1*, which gets passed as input to the VP. Finally, the output of the VP is *Out*, which is also the output of the whole sentence.

Then, in order to allow NPs to be missing, we need an NP rule that accepts no words, but instead uses the *who* that it received in its input:

```
np([who|Out],Out) --> [].
```

That is: One way to parse an NP is to accept *who* from the holding list rather than from the input string.

The remaining rules do not modify the holding list; they just pass it along un— changed from each step to the next. Rules for small constituents that cannot contain an NP do not need arguments, which is why *v* has no arguments here.

```
s(In,Out) --> np(In,Out1), vp(Out1,Out).
```

```
np(X,X) --> [max];[joe];[bill];[fido].
```

```
% Proper names are complete NPs
```

$vp(X, X) \dashv^* v.$

$vp(In, Out) \longrightarrow v, np(In, Out).$

$vp(In, Out) \dashrightarrow v, s(In, Out).$

$v \dashrightarrow [saw]; [said]; [thought]; [believed]; [barked].$

$v \dashrightarrow [see]; [say]; [think]; [believe]; [bark].$

'In a *ir/to*-question, this list will never have more than one member. The ability to stack more than one *who(m)* is needed for parsing nested relative clauses. For example:

The boy whom the girl whom we saw was liked ...

Holding lists go back to the work of Woods (1970), if not earlier, and are discussed in detail by Wanner and Maratsos (1978).

To parse a sentence, you must give the initial and final values of the holding list—namely

) and) —as extra arguments of `s` in the query, like this:

```
?- s([], [], [who, did, max, see], []).
yes
```

Holding lists are the basis of the “extraposition grammars” of Pereira (1981), who develops a useful extension of DCG notation for them. Their counterpart in unification-based grammar is “slash features” (Gazdar, Klein, Pullum, and Sag 1985).

Exercise 3.4.5.1

Take the parser just given, get it working, and show that it parses

Who did Max say thought Joe believed Fido barked?

Who did Max say Bill thought believed Fido barked?

Who did Max say Bill thought Joe believed barked?

but does not accept *Who did Max say Joe saw Fido?* or the like.

Exercise 3.4.5.2

Modify this parser so that, using another argument, it generates a tree structure. However, the tree should not show the actual order of the words; instead, it should show who in the position of the missing NP. For example, the tree for *I /to did Bill think said Fido barked?* should be:

NP

VP

NP

VP

NP

VP

Bill

think

who

said

Fido

barked

3.4.6 Separating Lexicon from PS Rules

It is often convenient to separate the descriptions of individual words (the rExiCON) from

```
noun(dog) .  
noun(cat) .  
noun(gardener) .
```

There are two advantages to doing this. First, because of indexing, it is much faster to search through the facts

```
noun(dog) .  
noun(cat) .
```

(etc.) than through rules of the form

```
n --> [dog] .  
n --> (cat) .
```

Second, and more importantly, words can be defined by rules as well as by facts. Here is a LEXICAL RULE that creates a noun ending in *-ness* from every adjective:

```
% noun(+N)  
% Strips the suffix "ness" from N and  
% looks for a corresponding adjective.
```

```
noun(N)  
    name(N,Nchars),  
    append(Achars, "ness",Nchars),  
    name(A,Achars),  
    adjective(A) .
```

```
% adjective(°A)
```

```
adjective(flat) .  
adjective(green) .  
adjective(blue) .
```

(Recall that `append` is not built in.)

This rule is not exactly right—it doesn’t change `y` to `i` when forming words such as `ugliness`—but it’s a start. The calls to `same` and `append` are time-consuming but unavoidable. In a complete working system, it might be better to modify the tokenizer so that some morphological analysis is done before the words are converted into atoms.

Exercise 3.4.6.1

Modify this lexical rule so that it changes final *y* to *i* before *-ness*. That is, the rule should produce correct spellings of words like *ugliness* and *slimmers*. Include a way to deal with exceptions such as *dryness* (not **driness*). Demonstrate that your implementation can correctly answer queries such as:

```
?- n([ugliness], [1]).
```

using the rule `add_endive(ugly)` plus the lexical rule.

Exercise 3.4.6.2

The lexical rule given so far expects the noun to be instantiated. That is, a query like `noun(11 a One s)` will succeed but `noun(X)` (with *X* uninstantiated) will fail. This means that the rule, as shown, is of no use in generating sentences. Rewrite it so that it will work with uninstantiated as well as instantiated arguments.

3.5 BUILDING SEMANTIC REPRESENTATIONS**3.5.1 Semantic Composition**

Syntactic structure is not the only kind of output that a parser can produce. For natural language understanding we also need a semantic representation—that is, a representation of meaning. DCG parsers can produce semantic representations too.

To demonstrate this, we'll work with a tiny subset of English in which the only noun phrases are proper names (Fido, Felix), thereby postponing some complicated questions about the semantics of NPs. The syntax that we will use is as follows:

S	$\rightarrow$	$NP VP$
NP		Fido
NP	--r	Felix
V	$\rightarrow$	chased
V		slept

We will represent the meanings of sentences in first-order predicate logic, so that *Fido chased Felix* will be *chased(fido, felix)* and *Felix slept* will be *slept(felix)*.

What about the meanings of individual words? Proper nouns are no problem, since they are logical individuals:

<i>Fido</i>	—	<i>fido</i>
<i>Felix</i>	—	<i>felix</i>

To represent verbs, we will use lambda expressions, just as we did with keyword systems

then

slept — (7x) slept(x)

where Xx indicates that the value of x is to be supplied from elsewhere.

So far, so good, but the verb *chased* needs two arguments, a subject and an object.

We will represent it with one lambda expression inside another:

chased — (7 y) Xx chased x, y)

This means, in effect, “Give me a value for y , such as *felix*, and I’ll give you another lambda expression that needs only a value for x , such as *ix*)chases x , *felix*).”

The next task is to combine the meanings of the individual words and thereby obtain the meanings of constituents. The parser will do this by supplying arguments to lambda expressions. For example, *fido* combines with Xx)*slept*(x) to give *slept fido*). Figure 3.6 shows how this works. Meaning seems to flow upward through the tree, as the meanings of smaller constituents get combined to give the meanings of larger constituents and ultimately of the whole S . But how can meaning flow upward when parsing proceeds top—down? The same way structures were built top—down, even though the actual information in them was only acquired at the bottommost (and therefore last) level. The parser will work with partly instantiated structures and instantiate the details when they finally become available.

Now for the implementation. Each predicate will have an argument for the semantic representation. We will represent lambda expressions in Prolog with the operator “ λ ”, just as in Chapter 2, so the rules for specific words will be:

`np(fido) --* lfido].`

`np(felix) --> [felix].`

`v(X'slept(X)) --> [slept].`

`v(Y' (X' chased(X,Y))) -- [chased].`

Then it is simple to write phrase-structure rules that combine their arguments in the desired ways:

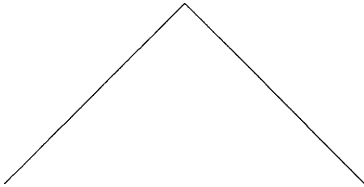
`s(Pred) --> np(Obj), vp(Obj Pred) .`

`vp(Obj' Pred) --> v(Obj' Pred) .`

`vp(Obj Pred) --> v(Obj' (Obj' Pred)), np(Obj) .`

S

chased fido, felix)

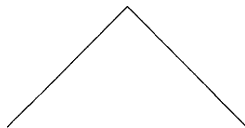


NP

fido

VP

Xx)chased x, felix)



$(X\ y)\ ix)chased\ x,\ y)$

felix

Fido

chas
ed

F
i
g
u
r
e
3
.
6

S
e
m
a
n
t
i
c
r
e
p
r
e
s
e
n
t
a
t
i
o
n
s
a
r
e
b
u
i
l
t
b
y
c
o
m
b

ining the meanings of the
Felix individual constituents.

The parser will then accept queries such as:

```
?- s(Semantics, [fido, chased, felix], [1]).  
Semantics = chased(fido, felix)
```

```
?- s(Semantics, [felix, slept], f]).  
Semantics = slept(felix)
```

Notice that the semantics also goes part of the way toward ensuring that *chased* has an object and *slept* does not. Unlike the original phrase-structure grammar, the parser does not accept **Fido slept Felix*, and although it accepts **Fido chased*, it produces a partly uninstantiated semantic representation that could easily be rejected by some other part of a natural language processing system.

This is a demonstration of sEMANTIC COMPOSITION, building the semantic representation of each constituent from those of its subconstituents. The technique shown here barely scratches the surface. We will return to semantic representations in Chapter 7, but first, there are many syntactic issues to be addressed.

Exercise 3.5.1.1

Get the parser working and add the words *saw*, *barked*, *Max*, and *Mary*. Generate semantic representations for *Fido barked* and *Mary saw Max*.

Exercise 3.5.1.3

Modify the parser described above so that it builds both syntactic and semantic representations. (Use one argument for each.)

3.5.2 Semantic Grammars

A SEMANTIC GRAMMAR is something intermediate between a keyword or template system and a phrase-structure grammar. Semantic grammars use phrase-structure rules, but words are classified by their function in a particular situation (such as computer commands or database queries) rather than general syntactic principles. Representations of meaning are built by any of several techniques, including the method described in the previous section.

Figure 3.7 shows some analyses of sentences that might be assigned by semantic grammars in various situations. There are no “right” or “wrong” analyses; the goal is

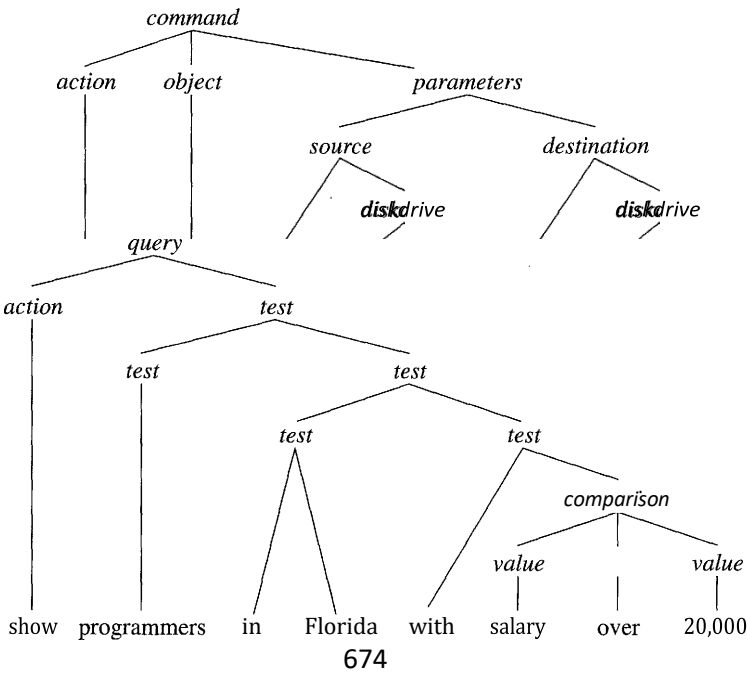


Figure 3.7 Semantic grammars assign arbitrary analyses to fit the purposes of a particular computer program.

purely to build something that works for a specific purpose. Parsing is often preceded by simplification just as in a template or keyword system.

Exercise 3.5.2.1

Reimplement your keyword system from Chapter 2 as a semantic grammar.

3.6 OFFBEAT USES FOR DCG RULES

Some people use DCG rules not just for parsing, but also to define almost any predicate that works through a list item by item. I do not advocate this practice, but you should be aware of it. Here is a predicate that counts the elements in a list:

```
count_off(N) --> [_], count_off(M), ( N is M+1 ).
count_off(0) --> [].
```

More precisely, this predicate accepts a specified number of elements from the beginning of the list, thus:

```
?- count_off(N, [a,b,c], [1]).
```

```
N = 3
```

```
?- count_off(N, [a,b,c,d,e,f], [e,f]).
```


$N = 4$

```
?- count_off(2, [a,b,c,d,e],What).  
What = [c,d,e]
```

Since this predicate has nothing to do with parsing, the use of DCG notation tends to obscure rather than clarify how it works, and `count_off` would be better off written in plain Prolog:

```
count_off(N, [_|Rest],Tail) :- count_off(M,Rest,Tail), N is M+1.  
count_off(0,Tail,Tail).
```

(Note that `count_off(0|_>_)` doesn't mean "do this at the end of the list"; it means "do this without accepting anything from the list," although in fact it is the only rule that can apply when the list is empty.)

Exercise 3.6.0.1

Use DCG rules to implement a predicate that accepts a list of numbers and computes their sum.

use it?

Exercise 3.6.0.3 (project)

Write a program that breaks a string of characters into words by using DCG rules. That is, it should convert "Lhis is iL" to Chi s , i s , i C) or the like.

This is a reasonable thing to do, because breaking a string into words is a kind of parsing. The phrase—structure rules might include the following, or something similar:

<i>Token</i>	→	<i>Special-character</i>
<i>Token</i>		<i>Alphanumerics</i>
<i>Alphanumeric</i>		<i>Alphanumeric—character</i>
<i>Alphanumeric</i>	→	<i>Alphanumerics</i>
<i>Alphanumeric-character</i>	→	a
<i>Alphanumeric-character</i>	→	a
<i>Alphanumeric-character</i>	→	b
<i>Alphanumeric-character</i>	→	
		c

For detailed advice see Appendix B.

3.7 EXCURSUS: TRANSITION-NETWORK PARSERS

DCG parsers are equivalent in power to an older parsing technique called AUGMENTED TRANSITION NETWORKS (ATNs). This section will briefly review ATNs as well as the simpler transition networks from which they are derived. There is little point in implementing an ATN in Prolog, since DCGs do the same job better, but it is useful to know how ATNs and DCGs correspond.

3.7.1 States and Transitions

A TRANSITION NETWORK is a parser that has a number of distinct STATES and proceeds

from state to state in a manner controlled by the input string.

Figure 3.8 shows a transition network that parses *the dog*, *the big dog*, *the cat*, and nothing else. Circles represent states and arcs represent transitions. A small arrow shows where to start, and a double circle indicates a state in which parsing can stop.

To parse a sentence, the parser must get from the initial state to a state in which it can stop. This is done by following the arcs and accepting, from the input string, the words that are on the arc labels. An unlabeled arc is called a PUMP ARC and allows a state transition without accepting any input. So the process by which this network parses *the dog* is:

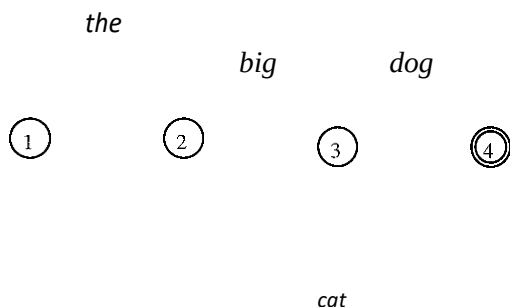


Figure 3.8 A finite—state transition network.'

Start in state 1.

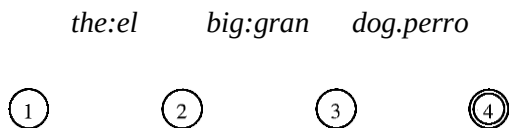
Go to state 2 accepting *the*. Go to state 3 via the jump arc. Go to state 4 accepting *dog*. Stop.

Because it has a finite number of states, this is a FINITE-STATE TRANSITION NETWORK (FSTN) or FINITE AUTOMATON.

A transition network is DETERMINISTIC if it cannot backtrack, and NONDETERMINISTIC if it can. More formally, a network is deterministic if, at every step, the choice of arcs is uniquely determined by the next word in the input string. In natural language processing, we work with networks in which nondeterminism is allowed, and we will assume that they are implemented in such a way that backtracking is possible.

A FINITE-STATE TRANSDUCER is an FSTN in which each of the transitions can produce output as well as accepting input. Figure 3.9 shows a finite—state transducer that can translate *the big dog* and *the cat* (but almost nothing else) into Spanish. The arc label *the:el* means “input *the* and output *el*,” and likewise for the other labels. Thus, this network translates *the dog* to *el perro*, *the big dog* to *el gran perro*; and *the cat* to *el*

gato. By reversing the roles of the input and output, we could just as easily get it to translate Spanish into English.



cat. gato

Figure 3.9 A finite-state transducer that translates English phrases into Spanish.

A finite—state transition network can contain c cLES; that is, an arc can loop back to the state that it started in, or even to an earlier state. This enables the network to

the big big dog

the big big big dog (etc.).

Although the number of states is finite, the number of words in the input string of this network is not bounded.

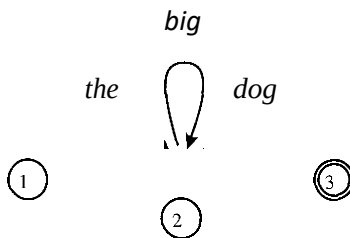


Figure 3.10 This finite-state transition network can accept arbitrarily long strings.

Finite-state transition networks are easy to implement in conventional programming languages, but they are clearly not adequate for human language; they provide no way to recognize constituent structure. Nonetheless, they have their uses in implementing templatelike systems (Chapter 2) and in analyzing morphology (Chapter 9).

Exercise 3.7.1.1

List the steps that the network in Figure 3.8 goes through when parsing *the cat*.

Exercise 5.7.1.2

Does the network in Figure 3.8 parse the *big cm*? Explain.

Exercise 3.7.1.3

Does the network in Figure 3.8 accept *the big* (with no words following)? Explain.

Exercise 3.7.1.4

Are the networks in Figures 3.8 and 3.10 deterministic?

Exercise 3.7.1.5

List the steps that the transducer in Figure 3.9 goes through when translating *the dog* into Spanish.

Exercise 5.7.1.6

Can the network in Figure 3.10 accept an infinitely long string? What is the difference between accepting an infinitely long string, and accepting arbitrarily long strings?

3.7.2 Recursive Transition Networks

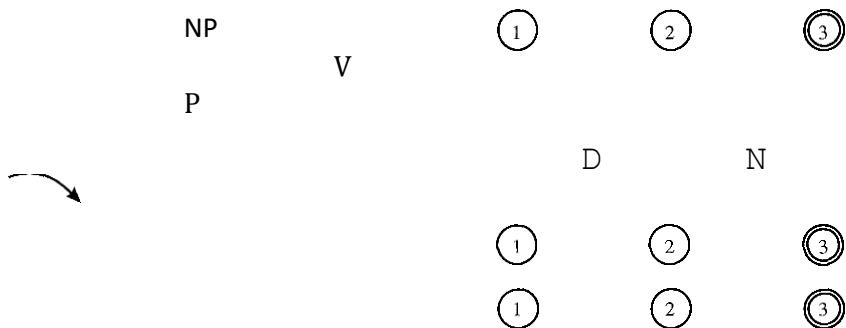
A RECURSIVE TRANSITION NETWORK (RTN) is one in which a state transition can either accept a word, or execute (call) another entire network. In particular, a network can call itself, either directly or indirectly (for example, S can call VP and VP can call S). This makes it possible to parse sentences within sentences. It also means that the number of states available to the parser is no longer finite, because any number of recursive invocations of the same network could be in use at the same time.

Figure 3.11 shows an RTN that parses sentences such as *The gardener said the butler thought the dog barked* (compare Section 3.1.3 above). By using recursion, this network parses an S within an S. Here the lexicon is shown separate from the transition network. This is standard practice, but the lexicon could, of course, be rendered as a transition network if we wanted to do so.

VP:

S:

NP:



Lexicon:

D: *the*

N: *dog, cat, butler, gardener*

V: *said, thought, barked*

Figure 3.11 A recursive transition network.

RTNs are essentially equivalent to phrase-structure rules or DCG rules without arguments. There is, however, a minor difference. RTNs can contain cycles that allow unlimited repetition (Fig. 3.12); DCG rules and ordinary PS rules cannot. This is not a serious problem, because any network containing a cycle can be replaced by a *set* of PS rules

which, of course, go into DCG straightforwardly.

Adj

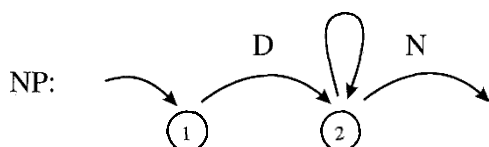


Figure 3.12 A cycle in an RTN.

These three rules generate the same sequences as the original network (D N, D Adj N, D Adj Adj N, etc.), but the tree structure is not the same. The rules introduce N' constituents which the original network does not. Fortunately, there are linguistic reasons for thinking that these N' constituents are real, and thus that the cycle in the network is not really appropriate for describing the structure of English.

Exercise 3.7.2.1

List the steps that the RTN in Figure 3.11 goes through when parsing each of these sentences:

The dog barked.

The butler said the gardener thought the dog barked.

Exercise 3.7.2.2

Construct a definite-clause grammar that is equivalent to the RTN in Figure 3.11.

Exercise 3.7.2.3

List the steps that the network in Figure 3.12 goes through when parsing *the big big big dog* (assuming an appropriate lexicon).

Exercise 3.7.2.4

3.7.3 Augmented Transition Networks (ATNs)

We saw earlier that the real power of DCGs comes from the ability to have arguments on the nodes. The equivalent of a DCG with arguments is an AUGMENTED TRANSITION NETWORK (ATN), which is like an RTN except that:

- Each subnetwork can have REGISTERS (memory locations) in which information can be stored.
- Each arc can have actions associated with it. These actions include storing, retrieving, and testing register values, and adding items to, or retrieving items from, a holding list.

- Data can be transferred between the registers of a subnetwork and the registers of the network from which it was called. For example, register values associated with a noun can be copied into the registers of the NP in which the noun occurs.

There are several different notations for ATNs, with different abstract instruction sets, and, compared to DCGs, all of them are cumbersome. Woods (1970) and Bates (1978) expound ATNs in detail. Pereira and Warren (1980) show that all ATNs can be translated into DCGs, and that, in general, the DCGs are more concise, more readable, and potentially faster to execute.

Figure 3.13 shows a simple ATN that enforces subject-verb agreement. Here are the steps that it goes through when parsing *The dog sees the* cols (ignoring backtracking):

Start in state 1 of S

network. Call NP network.

Start in state 1 of NP network.

Accept D (*the*).

Proceed to state 2 of NP network.

Accept N (*dog*, NAM—SINGULAR).

Set register NAM Of NP = SINGULAR.

Proceed to state 3 of NP network and exit.

Set register New Of S = SINGULAR.

Proceed to state 2 of S

network. Call VP network.

Start in state 1 of VP network.

Accept V *sees*, Num=SINGULAR).

Set register NAM Of VP = SINGULAR.

Proceed to state 2 of VP
network. Call NP network.

Start in state 1 of NP network.

Accept D *the*).

Proceed to state 2 of NP network.

Accept N *cats*, NUM PLURAL).

Set register NUM Of this NP = PLURAL.

Proceed to state 3 of NP network and exit.

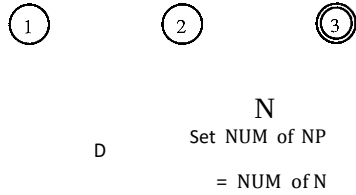
Proceed to state 3 of VP network and exit.

Test that NUM Of S = NAM of VP

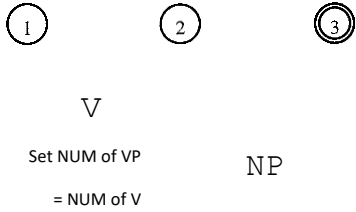
(the test succeeds; both have the value SINGULAR).

– NUM of NP = NUM of VP

S:



NP:



VP:



Lexicon:

D:

th
e

N:	<i>do g, cat, mouse</i>	NAM = SINGULAR
	<i>do gS, Cats, miCO</i>	NUM PLURAL
V:	<i>chases, sees</i>	NUM = SINGULAR
	<i>chase, see</i>	NAM = PLURAL

Figure 3.13 An augmented transition network that enforces subject-verb agreement.

Figure 3.14 shows an ATN that parses questions such as *Who did the gardener think chased the cat?* using a holding list.

A striking difference between ATNs and DCGs is that ATNs lack the concept of unification. Thus, in place of simply trying to instantiate variables, ATNs have to perform lots of separate assignments and comparisons. In Chapter 5 we will explore the usefulness of unification in grammatical analysis.

4.1 PHRASE STRUCTURE

The main purpose of this chapter is to develop a rough-and-ready set of phrase-structure rules for parsing some of the major structures of English, and, perhaps more importantly, to illustrate for a set of rules is developed.

Developing a formal grammar is a lot like writing a computer program; there are many points at which you can do any of several things as long as you follow up the consequences of your decision consistently. There are many other places at which one choice is probably better than another, but it will take a lot of research to find out which is better, and the difference may never show up during the lifetime of one project.

The rules developed here are not complete, for two reasons. First, if we tried to cover all the PS rules of English, this chapter would be hundreds of pages long.

Second, there are structures that cannot be parsed with phrase-structure rules alone; we will look at some of them and develop appropriate parsing mechanisms in the next chapter.

The rules in this chapter incorporate many of the ideas of Radford (1988), backenroff (1977), and Gazdar, Klein, Pullum, and Sag (1985) but do not adhere strictly to any theory. In particular, for convenience I ignore the distinction between adjuncts and complements, although I think it is valid; see Radford for a clear and detailed presentation.

4.1.1 Trees Revisited

Before developing PS rules we must look more deeply into their significance. Consider a syntactic tree like the one shown in Figure 4.1. The points where lines begin or end

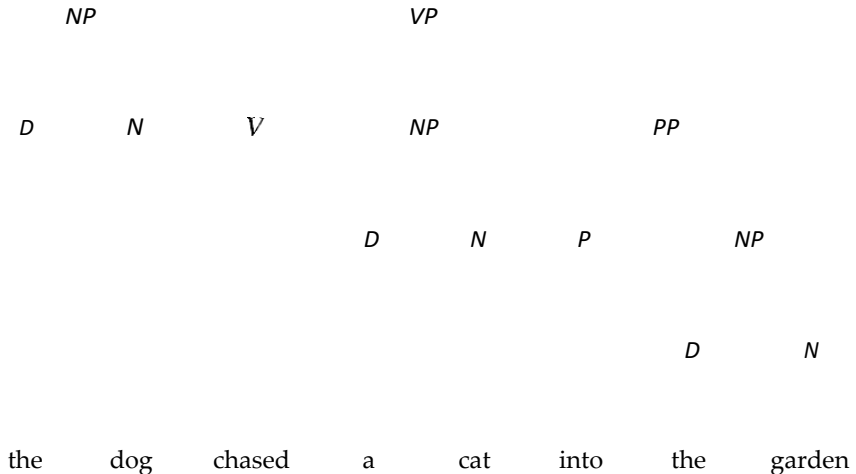


Figure 4.1 A syntactic tree.

in a syntactic tree are called NoDES. Normally, nodes bear LABELS Such as S, NP, VP, NP, or *dog*.

If two nodes are connected by a line, the upper node inMEDIATELY DOMINATES the lower one. More generally, one node DoMINATEs another if you can get from the first node to the second by following lines downward. For example, the first NP node dominates D, N, he, and *dog*.

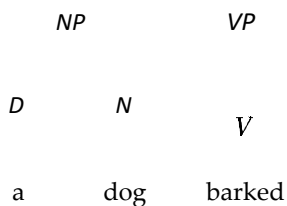
Two nodes are said to be sISTERS 1f they are immediately dominated by the same node. For example, D and N are sisters in the tree above. Their Mo+HER is NP; that is, they are nAuGHTERS of NP.

The words at the bottom of the tree diagram are the TERMINAL NODES. A CON-STITUENT consists Of all the terminal nodes dominated (immediately or indirectly) by a particular nonterminal node.

A tree is equivalent to flat LABELED BRACKETING of words into groups. For example, the tree shown at the top of page 79 could be written as

US SNP ID • DC IN *dog* NO w] [vP IV *barked* v) vP] S]

Without loss of information, the labels could be omitted on either the opening or closing brackets (not both). Labeled bracketings are often used to indicate part of the structure



of an example sentence, as in:

We though! f₅ it was raining).

Exercise 4.1.1.1

In Figure 4.1, which nodes are dominated by VP? Which are immediately dominated by VP?

Exercise 4.1.1.2

In Figure 4.1, identify the sisters of PP and the daughters of PP.

Exercise 4.1.1.3

Express Figure 4.1 as a labeled bracketing.

Exercise 4.1.1.4

Express $f_A(B [c xxx yyy]) [p zzz]]$ as a tree diagram.

4.1.2 Constituents and Categories

A tree gives two kinds of information about the sentence: it divides it into cONSTITUENTS (phrases) and classifies these constituents into cATEGORIES such as NP, VP, and the like. For example,

- *the* is a constituent of type D;
- *dog* is a constituent of type N;
- *the dog*, *a cat*, and *the garden* are constituents of type NP;
- *chased a cat into the garden* is a constituent of type VP;
- *the dog chased a cat into the garden* is a constituent of type S.

How do we know this is the right way to group the words in this sentence — that, for example, *into the garden* really is a constituent and a col *into* is not? Although none

a end. This argues strongly that info *the garden* is a constituent.

- « Any string of words that can be DELETED is probably a constituent. Again, you can leave out *into the garden* without changing the grammatical or semantic relations in the rest of the sentence. This, too, argues that *into the garden* is a constituent.
- Usually, the MEANING Of a constituent is, in some sense, a unit. It makes sense to ask what *into the garden* means; it makes much less sense to ask what a *cat into* means.

The strongest argument for the correctness of any phrase-structure tree, however, is the fact that it is generated by a coherent system of rules that also accounts for many other facts about English syntax.

The claims that a tree makes about categories are also important. For example, by labeling *the cat*, *the dog*, and *the garden* all with the label NP, this tree makes the claim that they are syntactically alike—they can all occur in the same positions. And indeed *a cat chased the dog into the garden*, or even *the garden chased a cat into the dog*, is a grammatical (if somewhat odd) sentence of English.

Exercise 4.1.2.1

List all the constituents that are identified by Figure 4.1.

Exercise 4.1.2.2

Cite at least two kinds of evidence that a *cat* is a constituent in the sentence *The dog chased a cat into the garden*.

Exercise 4.1.2.3

Cite evidence that, in *The dog t'hased a cat into the garden*, *the* and *n* belong to the same category.

4.1.3 Structural Ambiguity

Many sentences are AMBIGUOos, i.e., the same sentence can mean two or more different things. Usually the ambiguity is LEXICAL—that is, a word or idiomatic phrase within the sentence can have more than one meaning. For example, *glasses* can mean either spectacles or drinking—glasses.

But some ambiguities are sTRuCTURAL; they result from the existence of more

than one tree structure for the same string of words. Figure 4.2 shows a striking example: *I saw the boy with the telescope* can mean either that the boy had the telescope, or that I did the seeing with the telescope, and sure enough, the rules of English allow two tree structures, one with each meaning. In one tree, *the telescope* modifies *boy* (and thus *the boy with the telescope* is a constituent); in the other, *with the telescope* modifies *saw*.

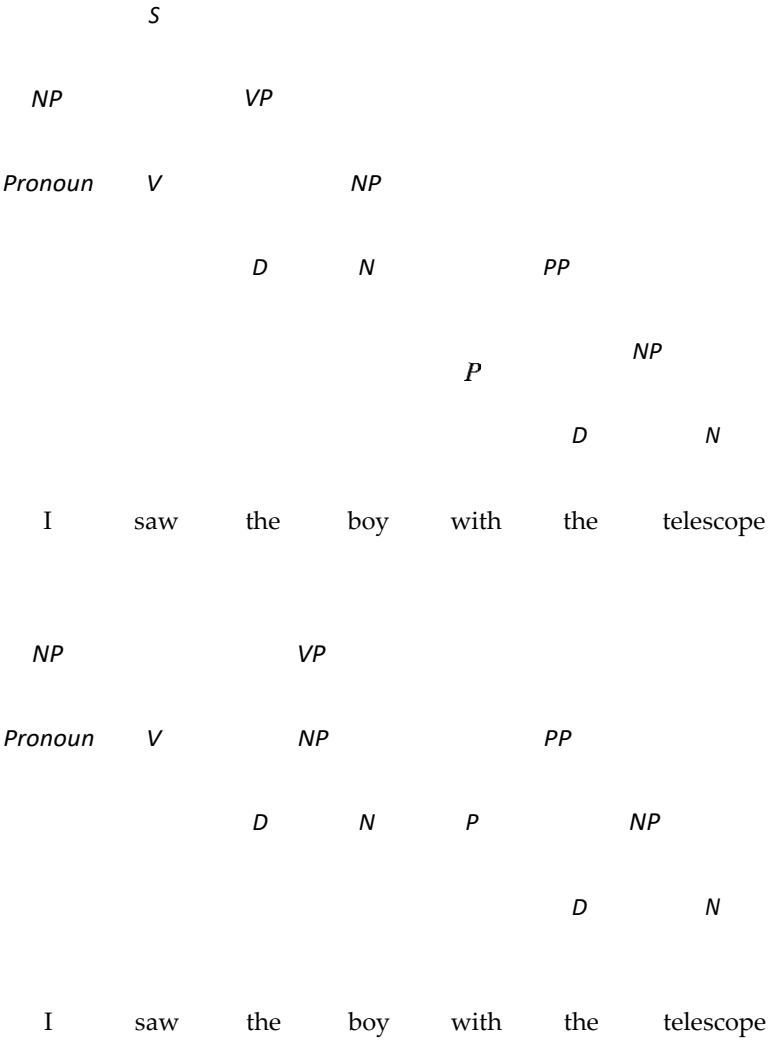


Figure 4.2 The two meanings of this sentence correspond to two tree structures.

Exercise 4.1.3.1

Give the two meanings of each of the following sentences, and state whether the ambiguity is structural, lexical, or both. Explain how you know.

By 1960 the Soviet Union had several satellites.

The painter put on another coat.

The judge threw the book at him.

Visiting relatives can be tiresome.

4.2.1 Parts of Speech

Many of the terms and concepts used in phrase-structure grammar come from a centuries- old tradition. The classification of words into “parts of speech” (nouns, verbs, adjectives, etc.) goes back to classical antiquity (Robins 1967) and was originally developed for Greek and Latin. It works reasonably well for English if some adjustments are made. This section will review traditional grammatical concepts briefly.

Table 4.1 illustrates the traditional parts of speech in English. The classification shown is neither complete nor entirely self-consistent. It is given mainly to illustrate the traditional terminology.

The first thing to note is that these categories are syntactic, not semantic. They are supposed to explain where each word can occur in the sentence; they are not a classification of word meanings. This is important because schoolbooks often say that “a noun denotes a person, place, or thing,” “a verb denotes an action or event,” and so forth. Such assertions are demonstrably false. Surely a sunset or a burglary is an event, but *sunset* and *burglary* are nouns, not verbs.

This does not mean that meaning is of no help in classifying words. Nouns *do* tend to refer to people, places, and concrete objects, and verbs *do* tend to refer to events, actions, or states. It’s just that this tendency is not absolute. It would be odd if *mountain* were a verb or if *kick* were not a verb.’

Even when words of different categories refer to the same thing, the meaning is packaged differently. The noun *burglary* and the verb *burglarize* denote the same kind of event, but there is a difference. The verb is marked for tense (past or present) and requires a subject and object; the noun does not. The noun, on the other hand, requires a determiner and can be made plural to denote more than one event of the same kind. In the Middle Ages, these differences in “packaging” were called *modes of signifying* (Covington 1984).

Exercise 4.2.1.1

Using traditional terminology, give the category of each word in each of the following sentences:

Syntax and semantics are my favorite subjects.

We found him easily when he was reading.

I feel sick.

Exercise 4.2.1.2

A few grammar handbooks say that the sentence *Hopefully, we’ll succeed* either is ungrammatical, or means ‘We will succeed in a hopeful manner’ (not ‘We hope we’ll succeed’ as the speaker intends). Based on what Table 4.1 tells you about adverbs, critique this claim.

But *mountain* is a verb in Nootka, a Native American language in which almost every word has both noun and verb forms. English, like most European languages, tends to make other categories into nouns; we have nouns derived from verbs (*destroy* → *destruction*), adjectives (*red* *redness*), and even particles (*out*

--r *outing*).

TABLE 4.1 TRADITIONAL SYNTACTIC CATEGORIES (PARTS OF SPEECH)

Nouns
Proper (name): <i>Joe, Frankenstein, America</i>
Common (ordinary)
Count (distinguishes singular from plural): <i>dog7dogs, theory7theories</i>
Mass (Noncount) (no plural): <i>water, art, stuff, superiority</i>
Gerund or verb : <i>singing, painting</i> (as in <i>painting is fun</i>)
Pronouns
Personal: <i>heNim, she/her, it, weans, you, the y7them</i>
Reflexive : <i>himself, herself, itsel/, ourselves .</i>
Demonstrative (Deictic) : <i>this, that</i> (as in <i>This is ii</i>)
Indefinite : <i>someone, everybod y, nobod y, something</i>
.
Determiners
Articles : <i>the, a (an)</i>
Demonstrative (Deictic) : <i>this, that</i> (example: <i>this house</i>)
Quantifiers : <i>even», all, some, three</i>
Possessive pronouns: <i>my, your, his, her, their</i>
Adjectives
Positive: <i>big, good, exceptional</i>
Comparative: <i>bigger, better</i>
Superlative : <i>biggest, best</i>
Participle of verb: <i>distracted, singing</i> (as in n <i>singing</i> cowboy)
Verbs

Intransitive (taking no object): *sleep, yell, bark*

Transitive (taking one object): *kick, emulate, destroy, read*

Ditransitive (taking two objects): *give* (as in *give Joe the book*)

Copula (verb of being): *be, am, is, are, was, were*

Modals (preceding another verb): *may, might, can, could, will, would . .*

Auxiliaries (preceding the verb, can follow a modal): *be, is, are, has*

Adverbs

Modifying a verb: *quickly, slowly, today, here*

Modifying an adjective or adverb: *very, extremely, slightly*

Modifying the whole sentence: *lioyeJ'ully, unfortunately*

Prepositions

With objects: *in, beJ'ore, after, below . .* (example: *ion the house*)

Without objects (Particles): *up, down, in, out* (example: *Look it up*)

Conjunctions

Coordinating (joining constituents of same type)

Simple: *and, or*

Correlating: *both . . . and, either . . . or, neither..... nor*

Subordinating (joining embedded sentence to main sentence):

before, afier, although, because, when, whenever

Traditional grammar analyzes sentences, not by drawing constituency trees, but by identifying relationships that connect one word to another. For example, in *The dog chased the cat into the garden*, a traditional grammarian would say that:

- dog* is the suBIECT Of *chased*;
- car* is the (DiRECT) OBJECT Of *chased*,
- into the garden* MoDIFiEs (describes) *chased*,
- garden* is the object of the preposition *into*, and
- the three occurrences of *the* modify *dog*, *car*, and *garden* respectively;

Table 4.2 lists the traditional names for grammatical relations. As you can see, there is often some uncertainty whether a grammatical relation belongs to a word or to a whole phrase; some people say the object of *chased* is *car* and others say it is *the cal*. In traditional grammar this uncertainty was never entirely cleared up.

TABLE 4.2 GRAMMATICAL RELATIONS IN TRADITIONAL GRAMMAR

Subject: Noun phrase required to precede a verb. Examples: <i>Birds fly</i> . <i>All the students are listening</i> .
Clause (embedded sentence) as subject: <i>That he succeeded is amazing</i> .
Predicate: The entire verb phrase. Examples: <i>Birds @y</i> . <i>All th students are playing soccer</i> .
Object: Noun phrase required to follow a verb or preposition. Object of preposition: <i>in the house</i> Object of verb: <i>John loves Mary</i> . <i>Asimov wrote two hundred books</i> . Indirect (first) object of two-object verb: <i>John gave Bill the answers</i> . Direct (second) object of two-object verb: <i>lohn gave Bill the answers</i> .
Complement: Something other than an NP required after a verb. Adjective as complement: <i>He looked silly</i> . Clause (embedded sentence) as complement: <i>We thought he was craa y</i> .
Modifier: Any word or phrase that describes another. Adjective modifying noun: <i>big dogs</i>

Adverb modifying verb: *barks loudly*

Clause modifying sentence: *When the y sang, we laughed.*

Grammatical relations can be defined in terms of phrase structure. For example, an object is an NP immediately dominated by VP or PP; a subject is an NP immediately dominated by S; and so forth (Chomsky 1965).

Exercise 4.2.2.1

Using Table 4.2 as a guide, identify as many grammatical relations as possible in the following sentences:

Three snails crept into the garden.

The slimy creatures gave us a sudden surprise.

When we saw them, we jumped.

4.3 THE NOUN PHRASE AND ITS MODIFIERS

4.3.1 Simple NPs

In constructing a set of phrase-structure rules we will begin with the NP. To a first approximation, the NP rule looks something like this:

$$NP \rightarrow D \text{ Adj} N (PP)$$

This accounts for NPs such as:

the dog

the gray cat

the dog in the garden

the young boy with the telescope

The determiner can be null, as in our well-worn example $t_N \text{ Birds}]_{Qy}$. This is accounted for by the rule

$$D \rightarrow \emptyset$$

alongside $D \rightarrow \text{rhc}$ and the like.

Most determiners are single words *the, a, some, every, five*, and the like). For now, we'll ignore NPs in which an article and a quantifier occur together (e.g. ID *the two boys*), as well as the internal grammar of numbers such as *three thousand four hundred twenty-two* and complex quantifiers such as *more than three*. For interesting analyses of phrases such as these, see Jackendoff (1977).

In Chapter 3 we saw that a determiner can consist of a complete NP followed by possessive *s*, as in tNP *the junkman*] *s daughter*. The NP can be of almost any type, but there seems to be a requirement that it end in a noun; you can say *the queen of England's crown* but not **the boy w/to ran quickly's prize*. We can assume that this requirement is a matter of morphology (the *'s* ending only goes on nouns) and thus that the PS rules

introduce two more PS rules:

NP → *Pronoun*
NP →+ *NDme*

along with the appropriate lexical entries such as:

Pronoun → *I, me, you, he, him, she, the y . . .*
Name →-r *Joe, Bill, Jack, Mary, Fido, Felix .*

Exercise 4.3.1.1

Construct a parser that implements all the PS rules discussed in this section, except for possessives. Use it to parse the noun phrases *birds*, *the dog*, *the gray cat*, *the dog in the garden*, and *the young boy with the telescope*.

This parser must construct a parse tree as a Prolog structure. Your instructor will test it with queries such as:

```
?- np(Structure, [the,gray,cat], []).  
Structure = np(dlthe),adj(gray),n(cat)
```

Remember that there is nothing in DCG notation that corresponds to the parentheses indicating optionality in a PS rule. You will have to work out, and implement, all the alternative rules in full.²

In most of the subsequent exercises in this chapter you will add rules to this parser. It will ultimately contain over 100 DCG rules.

Debugging a parser: If your parser fails to accept a phrase, first check that your rules include all the necessary vocabulary. For example, if

```
?- np(Str, [the,young,boy,with,the,telescope], []).
```

fails, check that *the*, *young*, *boy*, *with*, and *telescope* are in your DCG rules.

Next try to parse parts of the phrase. For example, try

```
?- np(Str, [the,young,boy], []).
```

(which, according to the grammar, should also be a noun phrase) and

```
?- pp(Str,[with,the,telescope],[ ]).
```

This will help you localize the rules that you have written incorrectly.

'Sometimes you can render *VP V (NP)* into DCG as '`vp --> v , (np ; (3) .`' where `(up , (3)` means 'accept an NP or a null constituent.' But if you use this trick while building tree structure in the way described here, your trees will be littered with unmotivated null constituents.

Once the parse succeeds, you're not through; you must verify that it gives the right structure, and that if more than one parse is possible, all the structures are consistent with the grammar.

4.3.2 Multiple Adjective Positions

A noun can take an indefinite number of adjectives in front of it, as long as the meanings add up to something coherent. For example:

the big dog

the big green dog

the big hairy green dog

the big fat hairy green dog

the big noisy fat hairy green dog

This, then, calls for a recursive rule. But the complete NP is not recursive; the other parts of the NP (particularly the determiner) are never multiple.

The solution is to introduce another level of constituent structure between N and NP. Jackendoff calls it N, N', or N'; in Prolog we'll call it n1. Then in place of

$$NP \rightarrow D [Adj] N [PP]$$

we can write:

$$NP \rightarrow D N' [PP]$$

$$N' \rightarrow Adj N'$$

$$N' \rightarrow N$$

and get structures like the one in Figure 4.3.

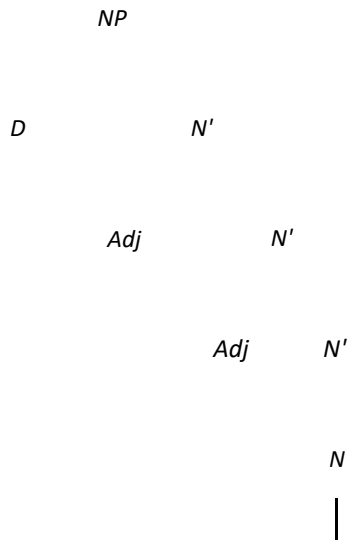


Figure 4.3 Recursion on the N' constituent allows NPs to have multiple

Revise your parser to allow multiple adjectives and parse *the big black noisy dOg*. It should still parse *the gray cat* and all the other examples from before, but now it should give structures that include N'

4.3.3 Adjective Phrases

An adjective is not always a single word—sometimes it is a phrase, such as *very big*. The N' rule introduces adjective *phrases*, not just adjectives:

the very big dog

the very big, surprisingly fat dog

the very big, slightly underfed, ounoyiugfy messy dog

So we need to change the first N' rule to

$$N' \rightarrow \text{AdjP } N'$$

and define adjective phrase (AdjP) as:

$$\text{AdjP} \rightarrow \text{r } (\text{Degree}) \text{ Adj}$$

where *Degree* $\rightarrow$ *very, slightly, extremely*, etc.

Longer AdjPs come after the noun, not before it, and AdjPs that follow the noun can include prepositional phrases and other complements after the adjective:

dOQ [AdjP similar to the first one]

We will not account for these here.

Exercise 4.3.3.1

Modify your parser to incorporate the rules just introduced, and to parse the examples given (the *very big dog*, etc.). It should still parse all the examples from previous exercises, but now the structures should include AdjP where applicable.

4.3.4 Sentences within NPs

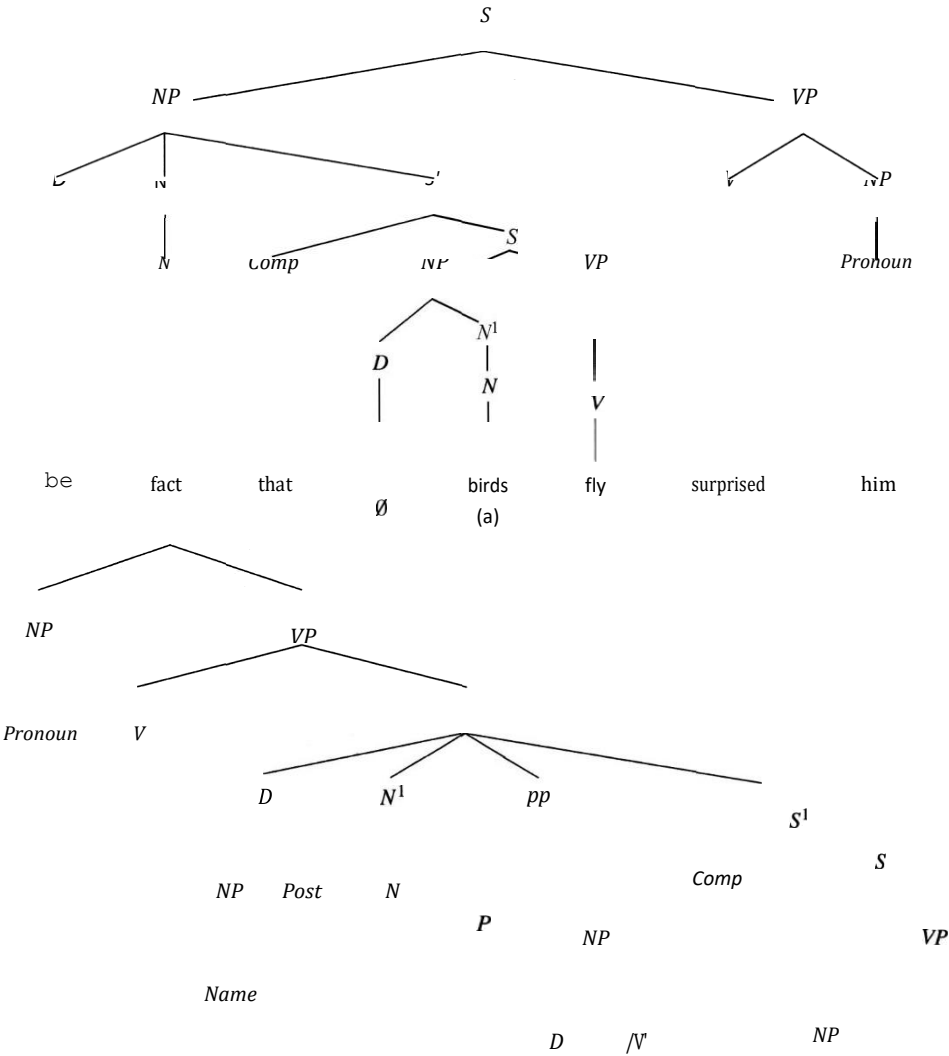
An NP such as *the fact that birds fly* has an S within it, preceded by the complementizer *that*. So we need to modify the NP rule again, and also define a new constituent S' (more commonly called S-bar, pronounced "S-bar"):

$$\begin{array}{lcl}
 NP & \rightarrow & D N^1 (PP) S' \\
 S' & & Comp S \\
 Comp & & \\
 & & \text{that}
 \end{array}$$

We also need to add the familiar rules:

$$V_P^S \Rightarrow \begin{matrix} NPVP \\ Y(NP) \end{matrix}$$

This will account for structures such as those in Figure 4.4.



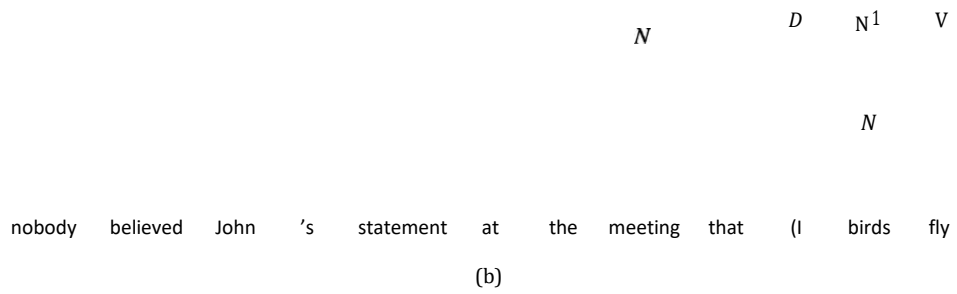


Figure 4.4 Sentences that contain complex noun phrases.

The fact that birds fly surprised him.

The students challenged the quite unexpected statement at the conference that birds fly.

Why do you get two parses for the second sentence? Is it genuinely ambiguous or is the grammar overgenerating?

1.4 THE VERB PHRASE

4.4.1 Verbs and Their Complements

Serbs take many kinds of complements after them. For example:

Verb	Type of complement(s)	Example
slept	None	<i>John slept.</i>
chased	NP	<i>The dog chased <u>the cat</u>.</i>
gave	NP + NP	<i>John gave us <u>the information</u>.</i>
gave	NP + [pp to .]	<i>John gave <u>the information to us</u>.</i>
said	S'	<i>John said [<u>that</u>] <u>birds fly</u>.</i>
seemed	AdjP	<i>John seemed <u>very old</u>.</i>
wanted	lo VP	<i>John wanted <u>to leave</u>.</i>

In Chapter 3 we looked briefly at the problem of SubCATEGORIZATION, i.e., ensuring that a verb is not used with the wrong kind of complement. For now, we will ignore subcategorization and simply assume that the input to the parser is grammatical.

To a first approximation, then, we need at least the following rules for VP:

VP	V (NP) (PP) (NP) (PP) (S')
VP	V AdjP
VP --r	V to VP

and we need to introduce a null complementizer:

Corny 8

This accounts for the structures in Figure 4.5, among others. But the first VP rule is far from satisfying, and we will return to subcategorization in Chapter 5.

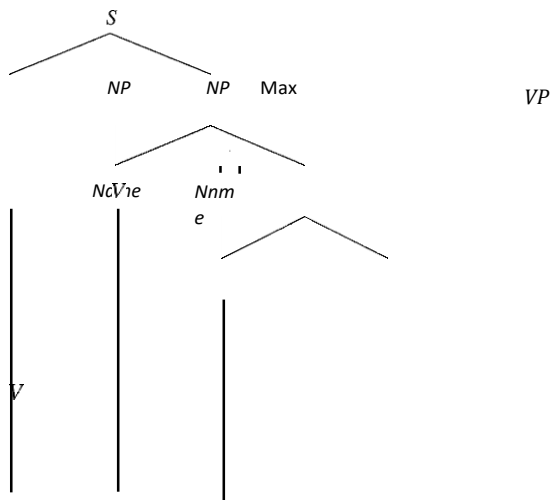
Exercise 4.4.1.1

Based on the PS rules introduced on this section, draw tree diagrams for the following sentences:

Birds flew into the garden.

Max announced at the meeting that the birds looked silly.

The birds wanted to leave.



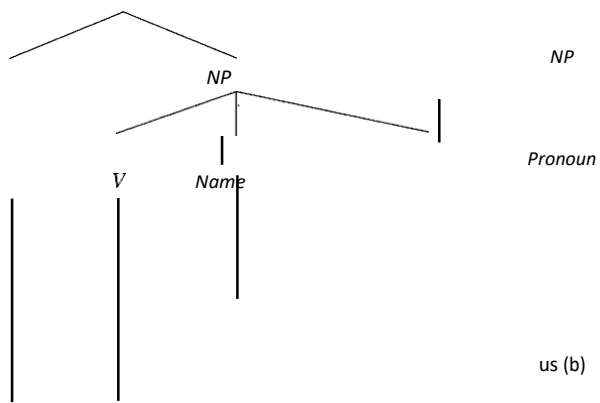
at

(a)

Max

spo
ke

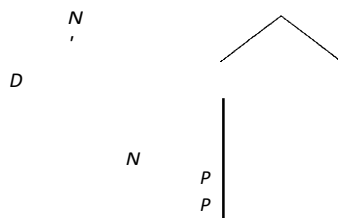
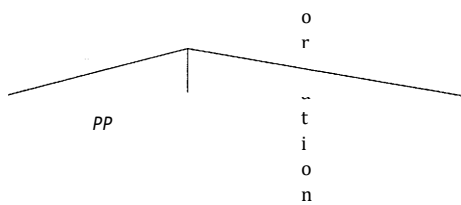
VP



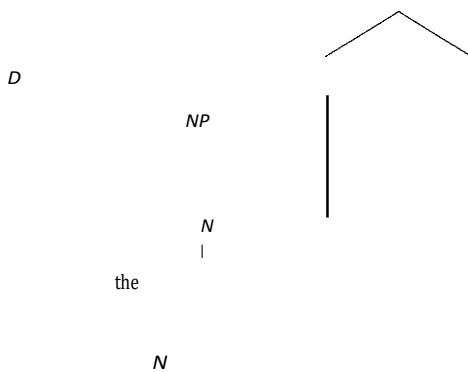
us (b)

VP

NP



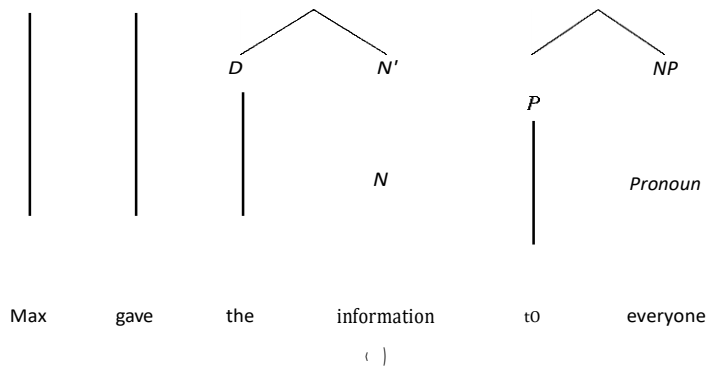
the meeting



the

N

i
n
f



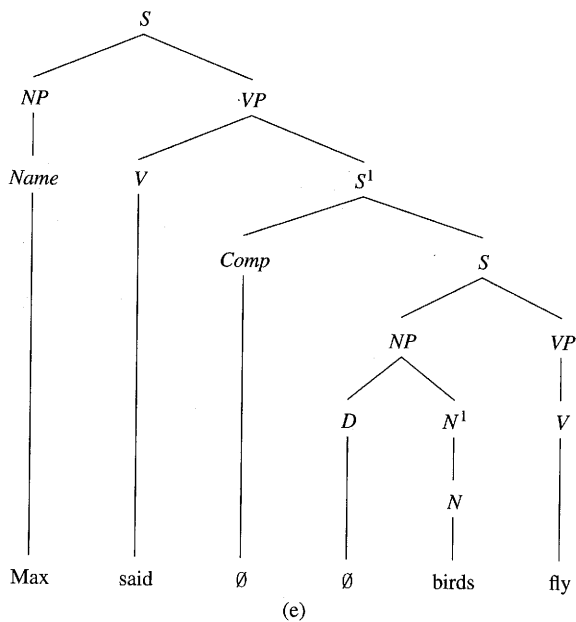
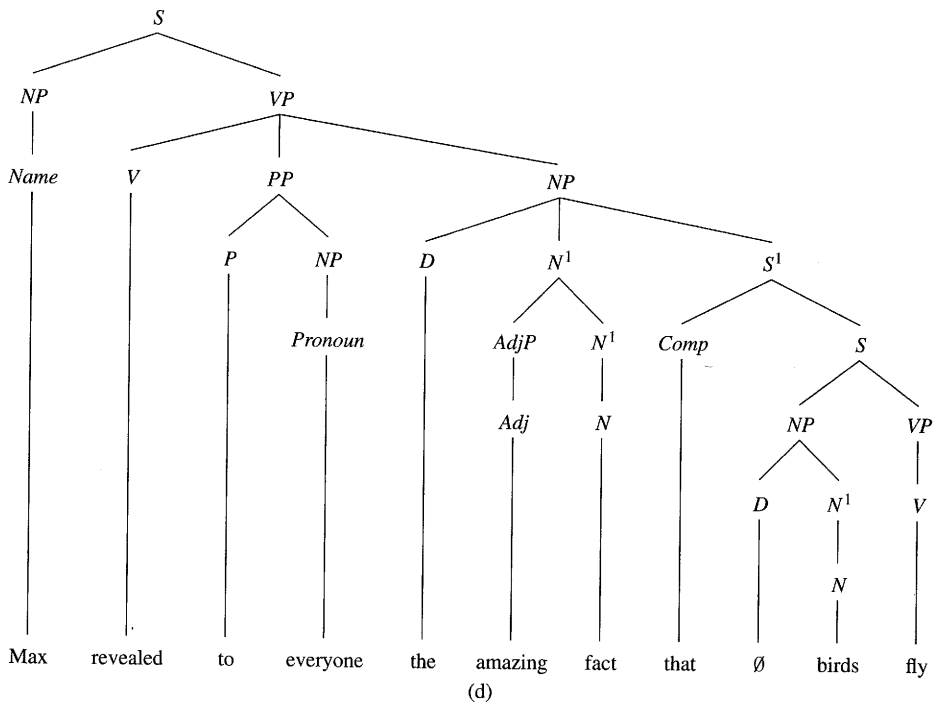


Figure 4.5 cont. (Continues on page 93.)

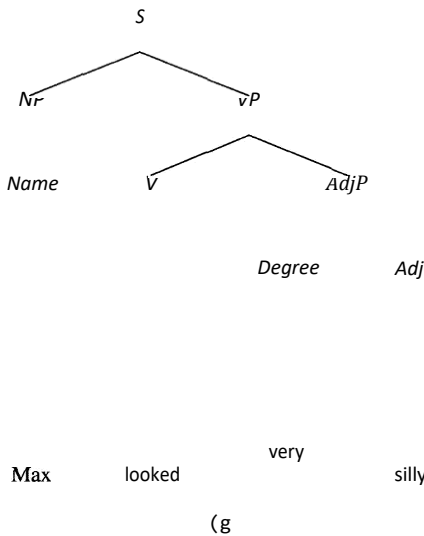


Figure 4.5 cont.

Exercise 4.4.1.2

Extend your parser to parse the sentences in Figure 4.5 and the sentences in the previous exercise. You need not include all expansions of the VP rule, as long as you include enough to parse the sentences here, and you are prepared to add more as needed later on.

4.4.2 Particles

A PARTICLE is a preposition without an object. Particles occur only with specific verbs which require them, such as *look up* or *throw out*. The same verbs also occur without the particles, with somewhat different meanings.

When present, the particle occurs in either of two positions, as shown in Figure 4.6 on page 94. Note that *Joe looked up the tower* is ambiguous (he either looked up the tower in a book, or looked upward along the tower), and that this ambiguity is structural; Figure 4.7 on pages 94 and 95 shows the two structures.

Exercise 4.4.2.1

Extend your parser to handle particles and to parse the sentences in Figures 4.6 and 4.7, giving both structures for *Joe looked up the tower*. You need not provide for all the combinations of particles with other parts of the VP; just add enough VP rules to parse the sentences needed for this exercise.

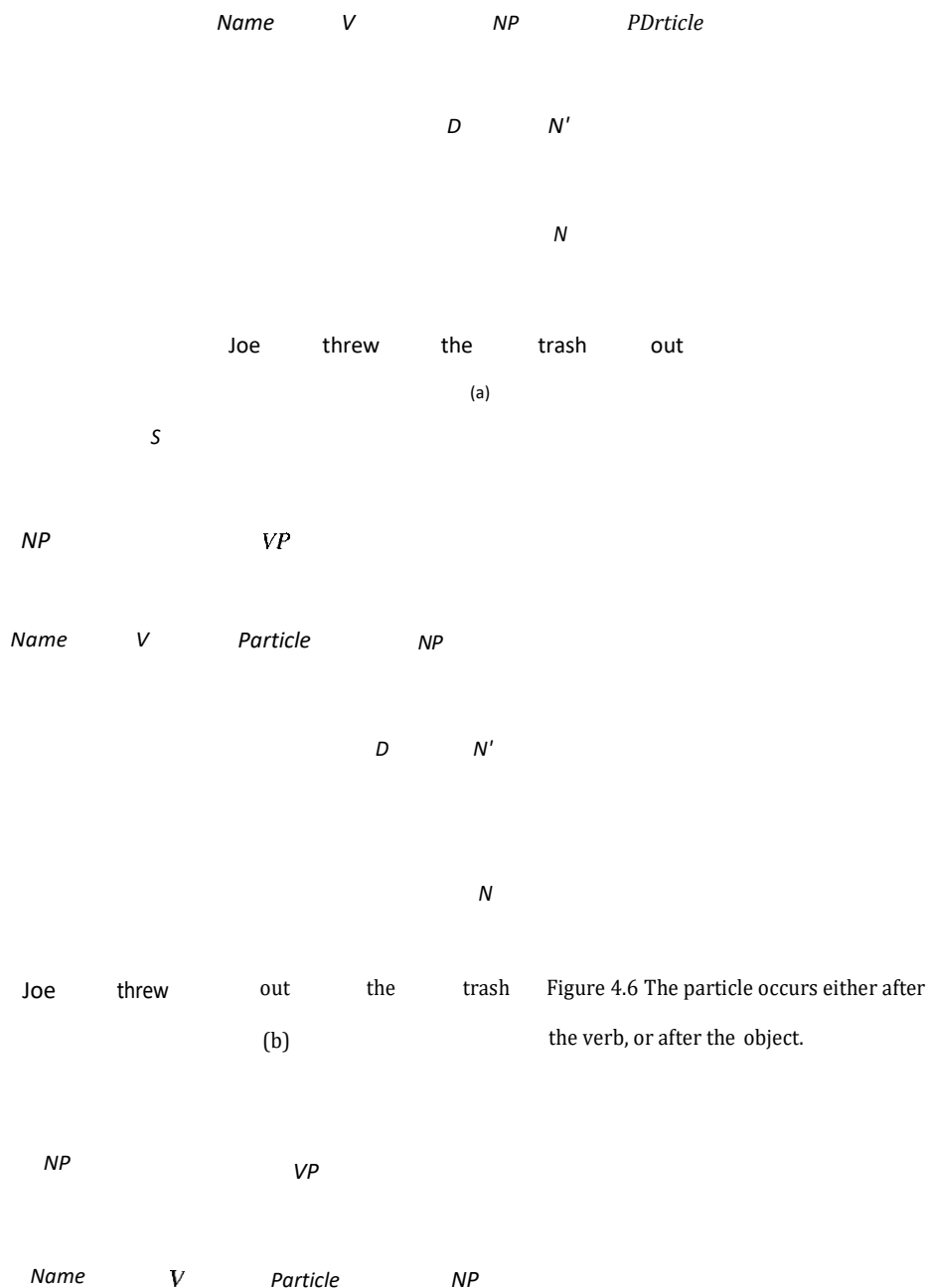
4.4.3 The Copula

The co SLA, or verb of being (*is, are, etc.*), takes an NP or AdjP as complement, as shown in Figure 4.8 on page 95.

Note (added 2008): In Fig. 4.8 we treat Copula as a separate syntactic category. There is a good case for treating the copula as a V with a particular subcategorization.

Exercise 4.4.3.1

Extend your parser to parse the sentences in Figure 4.8.



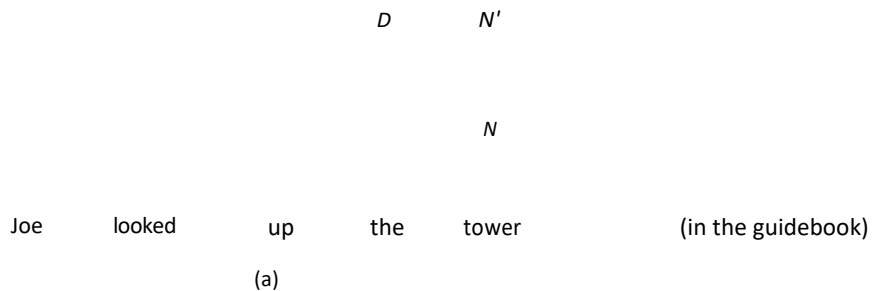
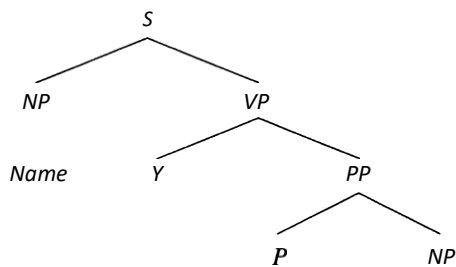


Figure 4.7 *Up* can be either an ordinary preposition or a particle. (Continues on page 95.)



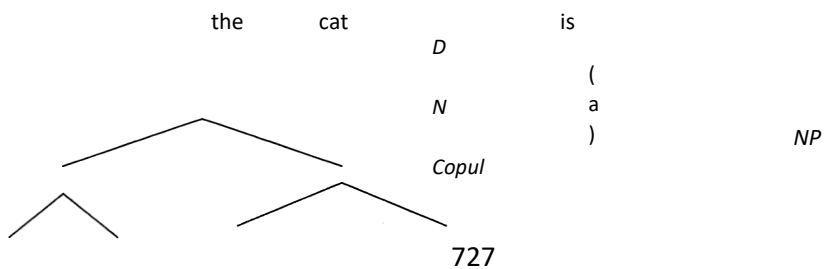
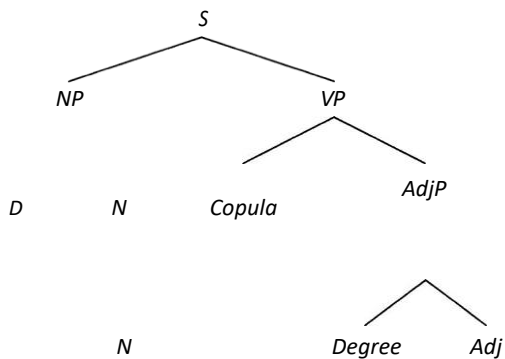
D *N'*

N

Joe looked up the tower (as he stood beneath it)

(b)

Figure4.7 conL



very

fat

N

D

N'



Adj

N

the

cat

is

my

favorite

pet

(b)

Figure 4.8 The copula takes an NP or AdjP as complement.

95

4.5.1 Conjunctions

Most occurrences of coordinating conjunctions such as *and* and *or* seem to be governed

$$X \rightarrow X \text{ Conj } X$$

by a rule

where X is any kind of constituent whatever. That is, any two constituents of the same kind can be joined by a conjunction to make a larger constituent. Figure 4.9 shows some examples.

The rule $X \rightarrow X \text{ Conj } X$ cannot be expressed directly in DCG notation; instead it must be replaced by a set of rules

NP *NP Conj NP*

AdjP *AdjP Conj AdjP*

NP

NP *Conj* *NP*

Name *Name*

Fido and Felix
(a)

V *Conj* *V*
hates detests
a
nd
(b)

AdjP

AdjP *Conj* *AdjP*

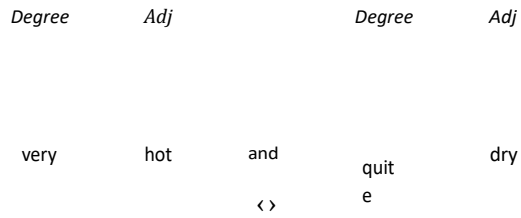
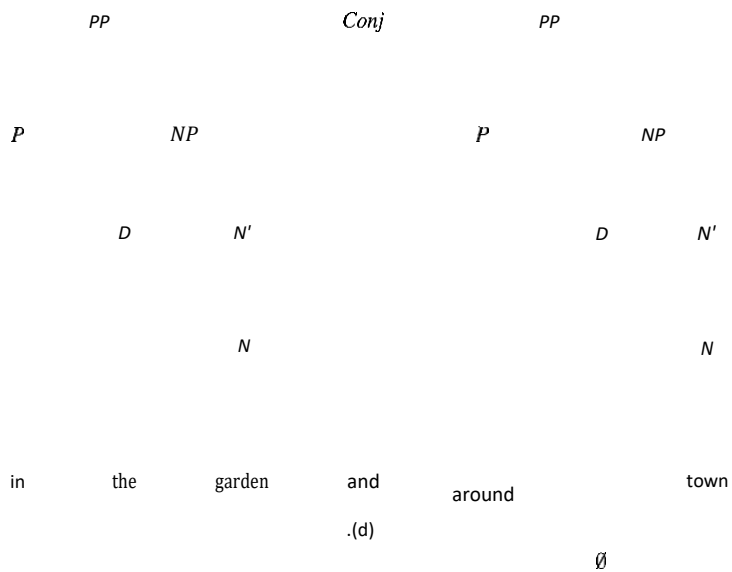


Figure 4.9 Conjunctions such as *and* take two constituents of the same kind and make them into a larger constituent.



VP

Conj

VP

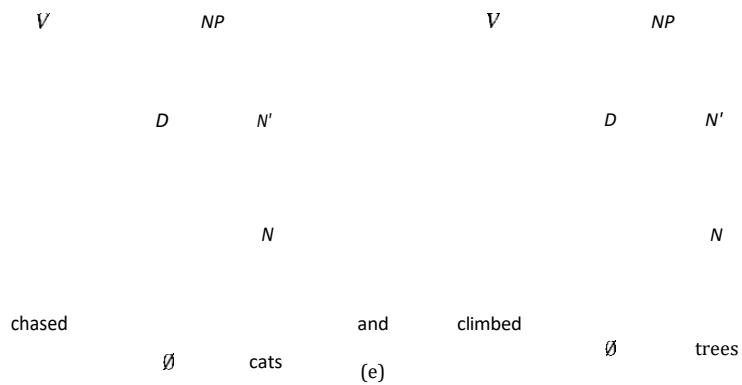


Figure 4.9 cont.

and so on. But even these rules aren't DCG passable; they cause loops, as noted in Chapter 3, where a trick for handling them was suggested.

Challenging as this is, it is still not the whole story about conjunctions. Hudson (1988) points out that in a sentence like

John drank [> coffee at breakfast) and (> tea at lunch].

the conjoined elements, *coffee at breakfast* and *tea at lunch*, are not constituents (Hudson 1988). Apparently, this sentence arises through a process of Erriesis (omission of understood material) from the complete sentence

f_s John drank coffee at breakfast) and [s John drank tea at lunch].

or at least the properly conjoined VP

J< we drank coffee at breakfast] attd vp drank tea at lunch].

There is no standard approach to parsing ellipsis phenomena such as these, but you should be aware of them.

Exercise 4.5.1.1

Draw a tree for *John drank coffee at breakfast and drank tea at lunch*. Now try to do the same thing with the second occurrence of *drank* omitted. What goes wrong?

Exercise 4.5.1.2

Why doesn't a loop arise when parsing correlating (discontinuous) conjunctions such as *both . . . and* and *cif/ter . . . on*

Exercise 4.5.1.3

Add rules to your parser to parse the following sentences:

It is both very warm and quite dry.
John drank both coffee and tea.

Do not attempt to handle any conjunctions other than *both . . . and*.

4.5.2 Sentential PPs

Traditional grammar recognizes numerous SUBORDINATING CONJUNCTIONS, such as *before*, *after*, *when*, *whenever*, and *because*, which allow a whole sentence to modify (describe) some part of another sentence. Examples:

I saw him after is he left].

The discussion after is he left] was surprising.

Here *after he left* modifies *saw* and *discussion* respectively.

Emonds (1976:172—176) and Radford (1988:134—137) argue convincingly that these “subordinating conjunctions” are not conjunctions at all, but rather *prepositions* of a special kind that take S rather than NP after them. Figure 4.10 shows the kind of structures that this entails.'

There are two main reasons to view “subordinating conjunctions” as prepositions. First, some of them *are* prepositions (of the kind we are already familiar with) and can equally well take an NP instead of an S. Examples:

after [s he left]

The discussion after fNP the meeting] w interesting.

before is he left]

I saw him before SNP the meeting]

Second, sentential PPs occur in the same positions as ordinary noun-phrase-containing PPs. We have just seen them at the ends of NPs and VPs. Both kinds of PPs also occur at the beginning of the sentence:

After he left

After the meeting the discussion continued.

This means that our familiar rule $S \rightarrow NP VP$ must be rewritten as:

$$S \rightarrow \text{--r } PP) NP VP$$

and the rules for PP become:

$$PP \rightarrow P NP$$

There is now a subcategorization problem with P because some P take only NP; some, such as *whenever*, take only S; and some take either NP or S. For the moment, we will ignore this.

³ For theoretical reasons the object of P may turn out to be not S but S with the null complementizer. In

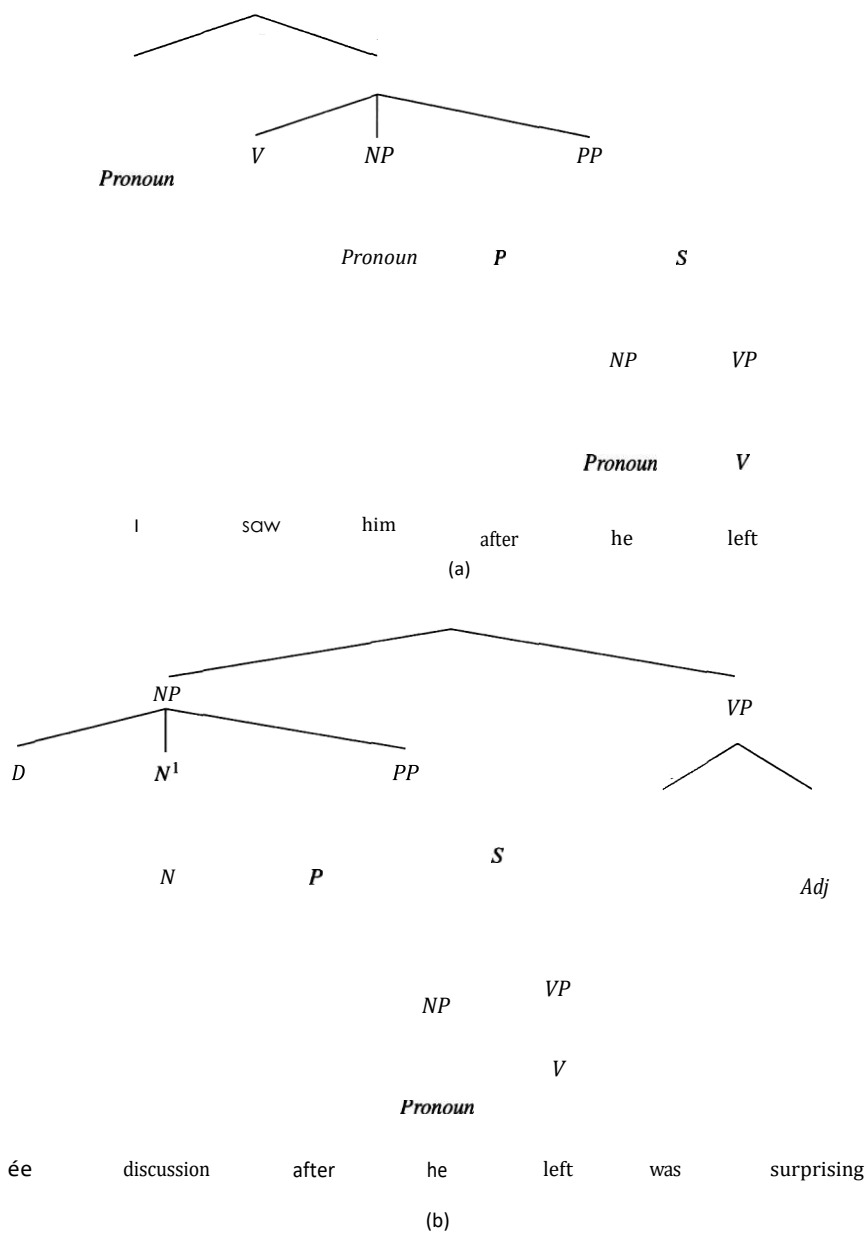


Figure 4.10 Examples of PP containing S.

Exercise 4.5.2.1

The sentence *I heard about the discussion after the meeting* is **structurally ambiguous**; *after the meeting* modifies either *discussion* or *heard*. Draw trees for its two structures.

Exercise 4.5.2.2

Extend your parser to handle the sentences in Figure 4.10, as well as:

After he left the discussion was surprising.
the meeting

Do not attempt to account for subcategorization of prepositions.

4.6 WHERE PS RULES FAIL

4.6.1 Adverbs and ID/LP Formalism

Like adjectives, adverbs take degree specifiers in front of them. The rule that accounts for this is

AdvP → *[Degree] Adv*

and the resulting structures include

AdvP *AdvP*
Adv *Degree* *Adv*

quickly very quickly

and the like.

The odd thing about AdvPs in English is the variety of positions in which they occur. For example:

Quickly he chased him into the garden.
He quickly chased him into the garden.
He chased him quickly into the garden.
He chased him into the garden quickly.

There is good evidence that the first two of these adverb positions hang from S and the latter two hang from VP, so that the structures are as shown in Figure 4.11.

This means that our S and VP rules need to be interspersed with optional AdvPs. Here is a stab at reformulating these rules:

$$\begin{aligned}
 S &\rightarrow \text{AdvP} [PP] NP (\text{AdvP}) VP \\
 VP &\rightarrow V (NP) \text{AdvP} PP (NP) [\text{AdvP}] (PP) (S') \quad (\text{highly dubious!})
 \end{aligned}$$

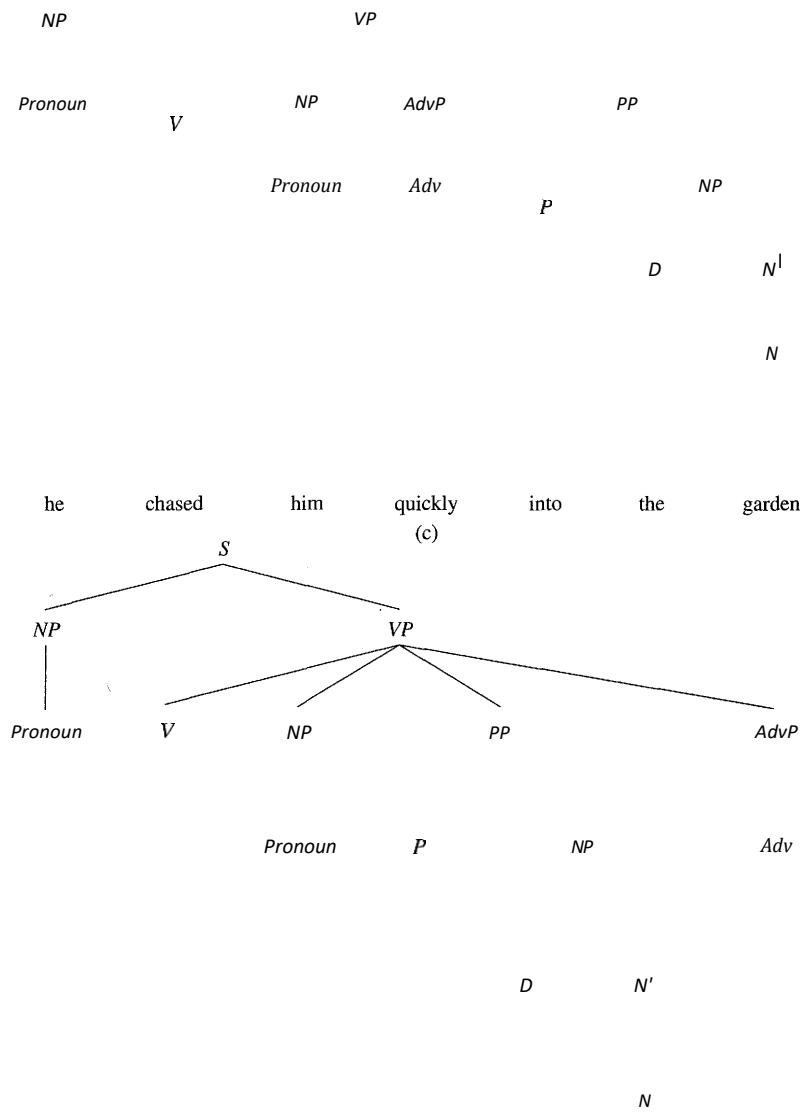
This is unsatisfying; the VP rule, in particular, is a real mess. Even though we've left out particles, the VP rule has seven optional elements and is therefore apparently the equivalent of 128 different DCG rules! In reality it's not quite that complex, because

<i>AdvP</i>	<i>NP</i>	<i>VP</i>				
<i>Adv</i>	<i>Pronoun</i>	<i>V</i>	<i>NP</i>	<i>PP</i>		
			<i>Pronoun</i>	<i>P</i>	<i>NP</i>	
					<i>D</i>	<i>N'</i>
						<i>N</i>
quickly	he	chase d	him	into	the	garden
<i>NP</i>	<i>AdvP</i>		<i>VP</i>			
<i>Pronoun</i>	<i>Adv</i>	<i>V</i>	<i>NP</i>	<i>PP</i>		
			<i>Pronoun</i>	<i>P</i>	<i>NP</i>	
					<i>D</i>	<i>N'</i>
						<i>N</i>
he	quickly	chase d	him	into	the	garden
			(b)			

Figure 4.11 Adverbs occur in many different positions in the English sentence.

it doesn't matter whether you choose the first NP or the second. Still, the whole thing is unwieldy.

What we would really like to say is that adverbs are a fundamentally different kind of thing than nouns or verbs. Instead of occupying fixed positions, AdvPs can go *anywhere* as long as they hang from an S or VP node and do not interrupt another constituent. That is, instead of having many *different* positions, the AdvP has an *unspecified* or "free" position.



he chased him into the garden quickly
(d)

Figure 4.11 cont.

But this is something that PS rules can't express. Phenomena like adverb placement led Gazdar, Klein, Pullum, and Sag (1985) and many others to replace PS rules with ID/LP RurEs. In ID/LP formalism, a rule such as

$$VP \rightarrow V, NP, PP, AdvP$$

an **ID (IMMEDIATE DOMINANCE)** rule. It doesn't say in what order the V, NP, PP, and AdvP occur. The order is established by one or more **LP (LINEAR PRECEDENCE) Rules** such as:

$$V < NP$$

$$NP < S'$$

which say that V precedes NP, V precedes PP, NP precedes S' , and so on (when they hang from the same node). This is only a partial specification of the ordering. Constituents can occur anywhere as long as they don't violate any LP rules. So if no position is specified for **AdvP**, AdvP can occur anywhere. IDfLP parsers have been developed (Kilbury 1984, Shieber 1984, Barton 1985, Leiss 1990).

Exercise 4.6.1.1

Extend your parser to handle all the sentences in **Figure 4.11**, plus the same sentences with *very quickly* in place of *quickly*. You need not add rules for expansions of VP that do not occur in these sentences.

Exercise 4.6.1.2

Convert the IDfLP rules

$$\begin{aligned} VP &\rightarrow V, NP, PP, AdvP \\ V &< NP \end{aligned}$$

$$NP < PP$$

into the complete set of equivalent PS rules.

4.6.2 Postposing of Long Constituents

There is a general tendency in English for long constituents to be **POSTPOSED** (placed at the very end of the sentence). This is obviously a practical thing to do; it lets the hearer parse as many constituents as possible, thereby obtaining context, before

tackling the longest one.

Here's an example. One reason our VP rule is so complicated is that we must parse both

Mix [revealed IN the fact] [pp Ot the meeting]].

and

MOX (vP revealed [pp Ot the meeting] (the amazing fact that birds fly]).

That is, we have both $VP \rightarrow V NP PP$ and $VP \rightarrow V PP NP$. The NP comes at the end if it is exceptionally long.

If we could explain in some other way why the long NP comes at the end of the sentence, we could simplify the VP rule. And indeed this seems like something we could do in ID/LP formalism: specify “the longest daughter of VP comes last” and let this take precedence over the other LP rules. So far, so good.

But there are cases where constituents are actually broken up in order to put a long constituent last. In transformational grammar this is called **ExTRAPoSITION FROM NP** (Radford 1988:448—456). Some examples:

A new book came out about the anatomy of dinosaurs. A problem arose that nobody expected.

John called people up who were from Boston.

Here *about the anatomy of dinosaurs* clearly modifies *book*, not *came out*, and the situation is analogous in the other two sentences.

The structures that we would like to assign these sentences are shown in Figure 4.12. The trouble is, these structures aren’t trees. They contain **DISCONTINUOUS CONSTITUENTS** that cannot be generated by PS rules.

The standard analysis of these sentences is to say that the PS rules generate them with the constituents unbroken, and another kind of rule, called a **TRANSFORMATION**, then moves a constituent to the end. On this analysis, the **nEE** (untransformed) structure of

A problem arose that I hadn’t foreseen.

is

SNP *A problem that I hadn’t foreseen] arose.*

and the postposing of *i6oi I hadn't foreseen* is a separate process.

To parse sentences with extraposed constituents, the parser will have to be ready, after parsing what appears to be a whole sentence, to pick up an additional constituent at the end and insert it in the proper place. The extraposition grammars of Pereira (1981) were designed partly to solve this problem. Extraposed constituents can also be handled by using features to pass information from one node to another as we did with *ali*-questions in Chapter 3.

Exercise 4.6.2.1

Suggest, in some detail, a way of parsing the sentences discussed in this section. You need not actually implement it.

4.6.3 Unbounded Movements

We saw in Chapter 3 that, in questions, the word *who* always appears at the beginning

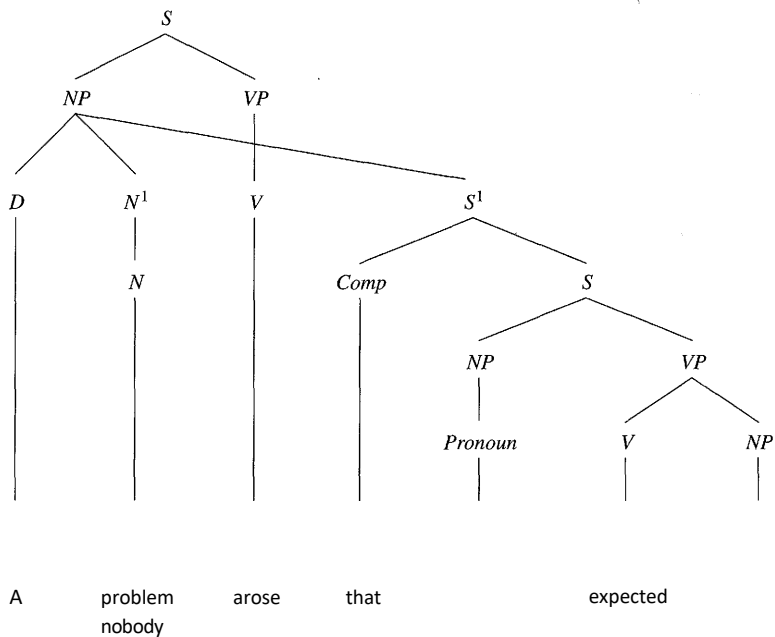
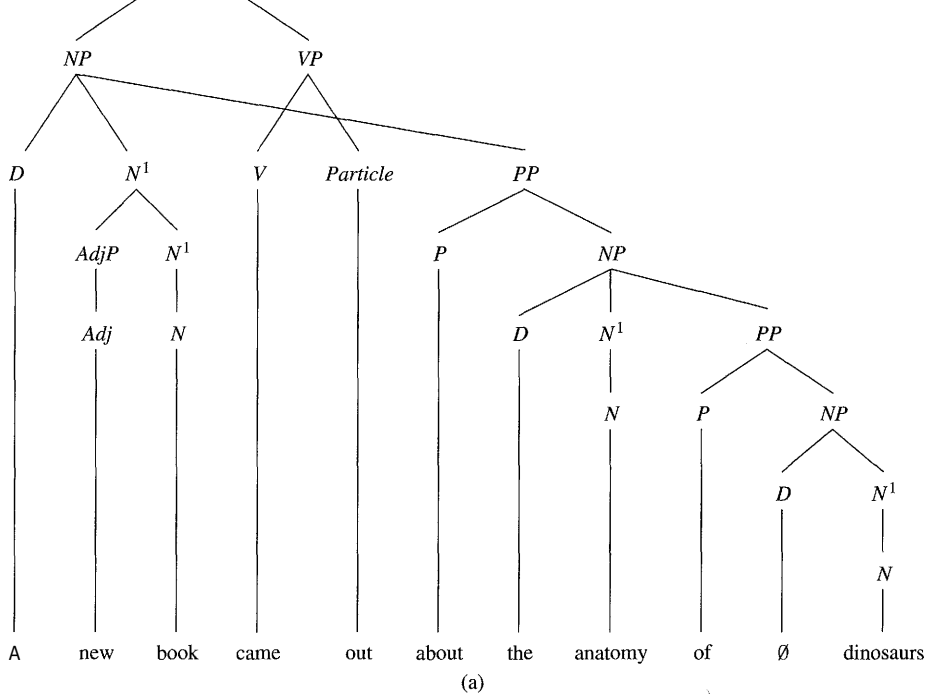


Figure 4.12 Long constituents are postposed, often breaking up the larger constituents in which they belong.

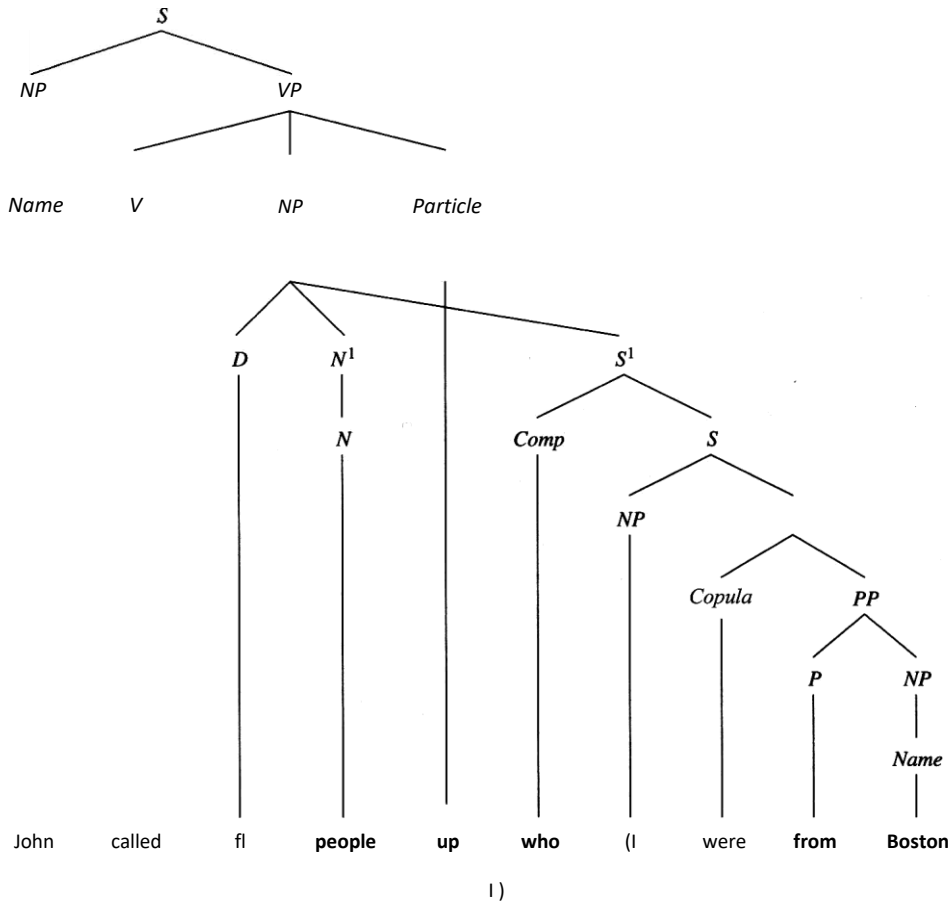


Figure 4.12 cont.

been moved from its original site to the beginning. Examples:

Who said Bill thought Joe believed Fido barked? (Max.) Who
did Max say u thought Joe believed Fido barked? (Bill.) Who
did Max say Bill thought believed Fido barked? [Joe.]
Who did Max say Bill thought Joe believed barked? (Fido.)

Here represents the missing NP.

This phenomenon, called wit-movement, occurs not only in questions, but also in exclamations such as

What a noise Max said Fido made z!

and in relative clauses (sentences modifying **NPs**) as in:

the boy who(m) Fido chased into the garden

But they cannot cross over each other; at any point, the parser can assume that the most recent wk-word will definitely correspond to the next missing NP.

WS-movement is an UNBOUNDED movement. This means that there can be any amount of structure between the original position of the moved word or phrase and the place it ends up. Thus PS rules cannot account for it. In Chapter 3 we parsed alt- questions with the aid of features; this is a standard approach (cf. the ‘slash features’ of Gazdar, Klein, Pullum, and Sag 1985).

Some structures are IsLANDS, which means that even unbounded movements cannot move material out of them. Conjoined structures of all types are islands. For example, even though

You saw Max and who(m)?

is grammatical (in a suitable context), it is not possible to perform alt—movement and get

** Who did you see Max and ?*

because box *and who(m)* is an island. Islands were discovered by Ross (1967).

Exercise 4.6.3.1

Look back at the feature-based parser for w/i-questions that we built in Chapter 3. Describe a way to add the rule

$NP \rightarrow \text{both } NP \text{ and } NP$

to this parser in such a way that $\text{SNP both } NP \text{ and } NP$] will be treated as an island.

4.6.4 Transformational Grammar

Rules that rearrange the structure of a tree are called TRANSFORMATIONS. They were introduced by Chomsky (1957), who experimented with PS rules and found them inadequate. Chomsky was the founder of modern generative grammar, and his introductory account of it is still worth reading.

Once transformations were introduced, linguists used them to account for all sorts of grammatical regularities, including:

- Agreement (Chapter 3) and case marking (Chapter 5), now handled with features.
- The relation between active and passive sentences such as:

The dog eats the food.

The food is eaten by the dog.

Nowadays these are accounted for by lexical rules (Chapter 9) which create, from every verb such as *cal*, an adjective such as *eaten* with the appropriate meaning.

- Various alternative word orders, such as *That he succeeded surprised me* versus *It surprised me that he succeeded*, which are now treated as alternatives in the PS rules.

Transformational grammar does not lend itself to parsing. The reason is that every transformation is a tree-to-tree mapping and thus cannot be undone without knowing the tree structure. So the parser has to determine the tree structure before undoing any transformations. And if transformations are necessary to account for tree structure, this is impossible.

In practice, transformational parsers rely on a *COVERING GRAMMAR*, a set of PS rules that account for the structures *after* the transformations have applied. But if the sentence can be parsed with the covering grammar, then there is usually no need to undo the transformations—the parser can proceed with other kinds of analysis immediately. Because of this, transformational grammar is seldom used in natural language processing.

Since the 1970s, the trend has been to replace transformations with more specialized mechanisms, such as features. Emonds (1976) gives a good summary of transformational grammar as it was in its heyday. In Chomsky's current theory, transformations remain as a means of accounting for certain movements, but they are defined in terms of more abstract principles, particularly *GOVERNMENT* (case assignment, Chapter 5) and *BINDING* (an abstract relation between specific positions in the sentence, such as the missing NP position and the moved word). For an introduction to government and binding theory, see Sells (1985).

Exercise 4.6.4.1

Implement, with DCG rules, a covering grammar for a grammar that contains the rules

<i>NP</i>	--r	<i>D N</i>
<i>VP</i>	→	<i>V (NP) (PP)</i>
<i>PP</i>	→	<i>P NP</i>
<i>D</i>	--r	the
<i>N</i>	→	dog, cat, garden
<i>V</i>	--+	slept, barked
<i>P</i>	--+	in

plus a transformation that optionally moves PP to the beginning of the sentence (as in *In the garden the dog barked*). Note that a covering grammar does not *undo* the transformation; it

There is no place you can go to look up “the rules of English” for computer implementation, because linguists do not yet agree on what form these rules should take. One of the most comprehensive modern generative grammars is that of Gazdar, Klein, Pullum, and Sag (1985); the classic study of phrase structure is Jackendoff (1977); and the best introductory textbook, for our purposes, is probably Radford (1988). The classic handbook by Stockwell, Schachter, and Partee (1973) relies so heavily on transformations that it does not lend itself well to parser implementation.

The best way to extend a parser is to feed it some actual text and see where it breaks down, then add rules as needed. For guidance on how to analyze particular syntactic phenomena, see Matthews (1981) on the nature of syntactic structure, and Huddleston (1988), and Quirk et al. (1973, 1985) for detailed descriptions of English. These are **DESCRIPTIVE** handbooks of grammar; they contrast sharply with **PRESCRIPTIVE** handbooks designed for teaching English to foreigners or teaching native speakers to write more clearly. Prescriptive handbooks are almost useless to the parser builder.

Among dictionaries, Hornby (1989) is especially useful because it specifies the kinds of complements required by each verb or noun.

5.1 A UNIFICATION-BASED FORMALISM

5.1.1 The Problem

In Chapter 3, we added FEATuREs (arguments) to the nodes in a phrase-structure grammar in order to do a lot of different things. We used features to account for agreement and case marking, build syntactic trees and semantic representations, and even undo movements. But we never put all these techniques together into a single grammar. Nor did we examine the role of features in contemporary linguistic theory. In this chapter we will do those things. In the process, we will develop an extension to Prolog that will make it much easier to use features in a grammar.

5.1.2 What is UBG?

A UNIFICATION-BASED GRAMMAR is any grammar that:

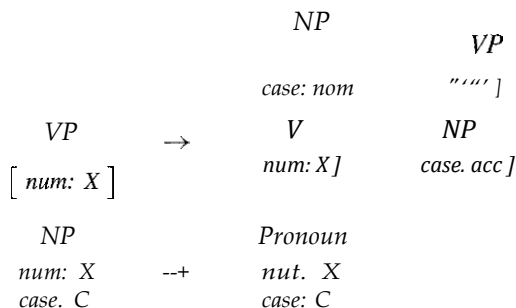
- encodes information in features and their values;
- gives values to features only through unification, and not through any other kind of computation.

look at features from a theoretical viewpoint.

Recall, for example, three rules from Chapter 3:

```
s --> np(Number,nominative), vp(Number).
vp(Number) --> v(Number), np(,accusative).
np(Number,Case) --> pronoun(Number,Case).
```

In unification-based grammar, the same rules are written:



Here every node except S has a FEATURE STRUCTURE, i.e., a set of features and values. The fact that S has no features is purely accidental; a more complete grammar would assign features to the S node as well.

Fig. 5.1 shows a syntactic tree annotated with features. Whenever a PS rule applies, the feature structures in the rule have to unify with the corresponding feature structures in the tree. For example, the first rule requires the NP and VP to have the same number and requires the NP to have *case: nom*.

Unification-based grammar is a relatively new development. Features are not; they go back to traditional grammar and are used freely by Chomsky (1965) and others. But unification did not appear on the linguistic scene until the 1980s.

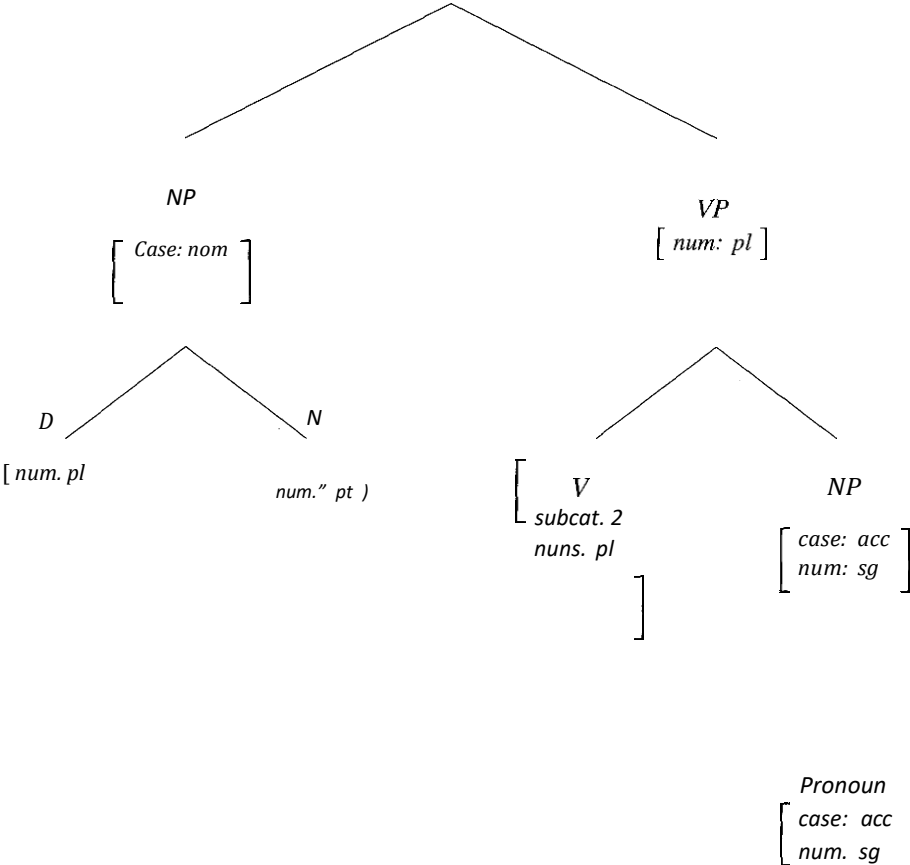
Further, UBG is not, in itself, a theory of grammar. Rather, it is a framework on which some (not all) theories of grammar are based, just as vector arithmetic is a framework for many theories in physics. Some theories of grammar that use various kinds of UBG are Functional Unification Grammar (FUG, Kay 1985, stemming from work in the late 1970s), Lexical-Functional Grammar (LFG, Bresnan 1982), and Generalized Phrase Structure Grammar (GPSG, Gazdar, Klein, Pullum and Sag 1985).

In this chapter we will develop a UBG formalism similar to that of Shieber (1985), but somewhat adapted to bring it closer to Prolog. We will then implement UBG in Prolog and use it to analyze a variety of phenomena in English.

5.1.3 How Features Behave

Features get where they are through several different processes, and although we are going to handle all of these processes through unification, we should start by

distinguishing them and giving them names.



The

dogs

scare

Figure 5.1 Example of a tree containing features.

Some features are properties of each word as listed in the dictionary. That is, the **LEXICAL ENTRY** Of a word supplies many of its features. Provisionally, we can think of a lexical entry as a simple PS rule such as:

Pronoun
 $\left[\begin{array}{l} \text{num. pl} \end{array} \right] \rightarrow \text{them}$

Some features are put in place by AGREEMENT RoLEs—rules that require a feature on one node to match a feature on another. For example, a plural noun requires a plural determiner (you can't say **one dogs*), and a singular noun requires a singular determiner. More concisely, the noun and the determiner Agree IN NUMBER. Here is a rule that makes them do so:

$$\begin{array}{ccc}
 NP & & D \quad N \\
 [\text{num}: X] & [\text{num}: X] & [\text{num}: X]
 \end{array}$$

This is, of course, just the PS rule $NP \rightarrow D NP$ with features added. Here the X in the feature structures is a variable, and this rule requires it to have the same value in the three places where it occurs.

Some features get into the tree by ASSIGNMENT; that is, the grammar requires them to have particular values. For example, the direct object of the verb has to have accusative case, so that *I see his* is grammatical and **I see he* is not. Here is a rule that assigns accusative case to the object:

$$\begin{array}{ccc}
 VP & \rightarrow & V \quad NP \\
 & & [\text{case}: \text{acc}]
 \end{array}$$

This is just $VP \rightarrow V NP$ with *case: rice* added in the right place.

Finally, some features get into place by PERCOLATION. That is, some of the features on a phrase are copies of the features of the main word in the phrase. For example, if the main verb in a VP is plural, then the VP itself is plural. The plural feature “percolates up” from the V to the VP . Here is the VP rule just given, but with percolation of the *num* feature added:

$$\begin{array}{ccc}
 VP & \rightarrow & V \quad NP \\
 [\text{num}: X] & & \text{num}: X \quad \text{case}: \text{acc}]
 \end{array}$$

Notice that the VP gets its number from the verb, not from the object.

Exercise 5.1.3.1

For each of the features in Figure 5.1, indicate whether the occurrence of the feature is best explained as coming from the lexical entry or from assignment, agreement, or percolation. Give justification for your claims.

5.1.4 Features and PS Rules

Before the 1980s, generative grammars treated agreement, percolation, and the like as processes performed by transformational rules. That is, the PS rules would generate the tree and then the transformations would copy features from place to place. Today, however, the leading view is that all features can be accounted for by just one operation— UNIFICATION—which applies along with each PS rule.

Recall that the purpose of each PS rule is to legitimize a particular part of the tree. For example, the rule

$$NP \rightarrow D N$$

legitimizes the structure:

NP

$$D \quad N$$

A grammar generates a tree if and only if every part of the tree is legitimized by some rule.

The features on each node in the tree have to be unified with the features on the corresponding node in the rule. For example, the rule

$$\begin{array}{ccc} NP & \rightarrow & D \quad N \\ \text{num: } X] & & [\text{num: } X] \quad [\text{num: } X] \end{array}$$

requires the structure $[\text{num: } X]$ to be unified with the feature structures of the NP, D, and N. Thus it ensures that the NP, D, and N have matching *num* features, and thereby accomplishes both agreement and percolation. The actual value of the *num* feature, singular or plural, is supplied in this case by one or more lexical entries. It could also have been supplied by some other rule assigning a *num* value to the whole NP.

Exercise 5.1.4.1

Using the NP rule just given, plus the lexical entries

$$D \rightarrow \text{the}$$

$$N \quad \text{dogs}$$

num: pl “

draw the complete tree for [pp *the dogs*].

5.1.5 Feature-Structure Unification

The features in a feature structure are identified only by name, not by position. Thus

<i>person: 2</i>		<i>number: plural</i>
<i>number. plural</i>	an	<i>person. 2</i>
	d	

are the same feature structure.

Two structures can be UNIFIED if they can be combined without contradiction. For example,

$a: b$ and $a: b$ unify to give $\begin{bmatrix} a. & b \\ c. & d \end{bmatrix}$

are simply left out. Thus there is no need for an “anonymous variable.” Variables with names, however, work the same way as in Prolog. For example,

$$\begin{array}{l} a: b \\ d \\ c: d \end{array} \quad \text{an} \quad \left[\begin{array}{l} a: X \\ e: X \end{array} \right] \quad \text{unify to give} \quad \left[\begin{array}{l} a: b \\ c: d \\ e: b \end{array} \right]$$

The second feature structure doesn’t give values for a and e , but it imposes a requirement that the values of a and e must be the same. This is a lot like what happens if you unify $f(b, d, p)$ with $f(X, p, X)$ in Prolog. As in Prolog, we will stipulate that

like-named variables are the same if and only if they occur in the same structure or the same rule.

Feature—structure unification can, of course, fail. In such a case the grammar rule requiring the unification also fails, i.e., cannot apply. For example, the feature structures

$$\left[\begin{array}{l} a: b \\ c: d \end{array} \right] \quad \text{an} \quad d \quad \left[\begin{array}{l} a: d \\ e: f \end{array} \right] \quad \text{do not unify}$$

(the unification fails) because a cannot have the values b and d simultaneously in the same feature structure.

A big advantage of unification—one that we’ve already exploited in Prolog—is that it is **ORDER-INDEPENDENT**. If you unify a set of feature structures, you’ll get the same result no matter what order you unify them in. This means that it is often possible to use a single unification-based grammar with many different parsing algorithms. It doesn’t matter which unifications get done first, as long as all the prescribed unifications are eventually performed. This gives great freedom to the programmer who is designing a parser.

The order-independence of unification also eliminates a vacuous question that arose in transformational grammar. Consider subject—verb agreement, for example. Does the number feature get copied from the subject onto the verb, or from the verb onto the subject? Obviously, it makes no difference. Yet a transformational grammar has to make the copying go in one particular direction; a unification-based grammar merely says that the number features of the subject and of the verb are equal.

Exercise 5.1.5.1

Unify the following feature structures, or indicate why the unification fails.

1. $\left[\begin{array}{l} \textit{number: sg} \\ \textit{case: acc} \end{array} \right]$ and $\left[\begin{array}{l} \textit{number: eg} \\ \textit{person: 3} \end{array} \right]$
2. $\left[\begin{array}{l} \textit{number: N} \\ \textit{case: nom} \end{array} \right]$ and $\left[\begin{array}{l} \textit{case: nom} \\ \textit{person: 3} \\ \textit{number: pl} \end{array} \right]$

3. $\begin{bmatrix} \text{case: } C \\ \text{person: } P \\ \text{number: } eg \end{bmatrix}$ and $\begin{bmatrix} \text{person: } 3 \\ \text{number: } pl \\ \text{case: } acc \end{bmatrix}$

4. $\begin{bmatrix} a. \ b \\ c. \ Y \\ e. \ N \end{bmatrix}$ and $\begin{bmatrix} a: \ Y \\ c. \ X \\ e. \ X \end{bmatrix}$

5. $\begin{bmatrix} n: \ h \\ c: \ Y \\ e. \ f \end{bmatrix}$ and $\begin{bmatrix} a. \ Y \\ c. \ d \\ e: \ X \end{bmatrix}$

Exercise 5.1.5.2

Here are three feature structures:

$\begin{bmatrix} a. \ b \\ c: \ X \end{bmatrix}$ $\llcorner$ $\begin{matrix} a: \ Z \\ e: \ Z \end{matrix}$

$\uparrow$

$e: \ Y$

Unify the first (leftmost) structure with the second and show the result; then unify that with the third. Then do the same thing again, taking the structures in the opposite order (right to left).

5.2 A SAMPLE GRAMMAR

5.2.1 Overview

Now it's time to build a real, working unification-based grammar (working in the sense that the rules will fit together properly and generate sentences; we won't put it

on the computer just yet). This grammar will be based on the PS rules:

<i>S</i>	→	<i>NP VP</i>
<i>VP</i>		<i>V (NP)</i>
<i>NP</i>	→	<i>Pronoun</i>
<i>NP</i>	→	<i>D N</i>
<i>Pronoun</i>		he, him, it, they, them
<i>D</i>	→	the, a, two
<i>N</i>		dog, dogs, cat, cats
<i>Y</i>		bark, barks, scare,

scares To this we will add features to enforce five constraints:

- Number agreement of subject and verb;

- Assignment of accusative case to object;
- Subcategorization to distinguish verbs that do and do not take objects.

Exercise 5.2.1.1

Show why each of the five constraints just mentioned is needed. That is, for each constraint, give a sentence (with tree) that is generated by the PS rules but is ungrammatical because it violates the constraint.

5.2.2 Lexical Entries

First, the lexical entries. The pronouns are simple:

Pronoun

case: nom he

num: sg

Pronoun

case: acc → him

num: sg

Pronoun

[mm. *sg* it

Pronoun

case. nom they

num: pl

Pronoun

case. acc them

Notice that *it* has the same form in both nominative and accusative (you can say both *it scares him* and *he scares it*), we capture this fact by simply leaving out its *case* feature.

Now for the nouns. Ideally, we'd like to have a rule that adds *-s* to the singular form of each noun to make the plural. That will have to wait until Chapter 9; in the meantime we will simply list both the singular and the plural form of each noun:

N
num: sg] → dog

N
num: pl “ dogs

N		
$[\textit{num: sg}]$	$\rightarrow$	cat
N		
$\textit{num: pl}$		cats

Notice that nouns are not marked for case.

Determiners agree with nouns in number; that is, some determiners are singular and some are plural.

D		a
$\textit{num. sg}]$		
D		
$[\textit{num. pl}$		'

The goes with both singulars and plurals, so we leave its number feature unmarked:

D	--r	the

Next, the verbs. We use a feature called *subcat*(egorization) to distinguish verbs that take an object, such as *scare*, from verbs that don't. Like nouns, verbs are marked for number, but this time the -s is *absent* in the plural. We ignore person agreement (/ *scare*, *you scare* vs. *he scares*).

$\textit{num: sg}$	--+	barks
$\textit{subcat: 1}$		
$\textit{num. pl}$		bark
$\textit{subcat. 1}$		
$\textit{num. eg}$	--+	scare
		s

subcat: 2

num. pl --r scare
subcat: 2

Exercise 5.2.2.1

Write lexical entries (in the same form as those above) for *she*, *elephant*, *every*, *all*, *chase*, and *chases*.

5.2.3 Phrase-Structure Rules

The phrase-structure rules are simple. Consider first the rule:

NP *D* *N*

of view; in UBG it doesn't matter.)

The pronoun rule is even simpler, except that it has to percolate case as well as number:

$$\begin{array}{c} NP \\ \left[\begin{array}{l} \text{case: } C \\ \text{num: } X \end{array} \right] \end{array} \rightarrow \begin{array}{c} \text{Pronoun} \\ \left[\begin{array}{l} \text{case: } C \\ \text{num: } X \end{array} \right] \end{array}$$

There are two VP rules and the *subcat* feature determines which one any particular verb takes:

$$\left[\text{num: } X \right] \rightarrow \begin{array}{c} \text{subcat: } 1 \\ \left[\text{num: } X \right] \end{array} \quad (\text{for verbs without objects});$$

$$\begin{array}{c} VP \\ \text{num: } X \end{array} \rightarrow \begin{array}{c} NP \\ \text{case: acc} \\ V \\ \left[\begin{array}{l} \text{subcat: } 2 \\ \text{num: } X \end{array} \right] \end{array} \quad (\text{for verbs with objects}).$$

The second rule assigns accusative case to the object.

Finally, the S rule assigns nominative case to the subject and enforces subject-verb number agreement:

$$S \rightarrow \begin{array}{c} NP \\ \left[\begin{array}{l} \text{case: nom} \\ \text{num: } X \end{array} \right] \end{array} \left[\begin{array}{c} VP \\ \text{num: } X \end{array} \right]$$

In this grammar, the S node itself has no features. As mentioned earlier, this is purely accidental; a more complete grammar would give features to the S, but in this grammar there happen to be none.

Exercise 5.2.3.1

Add feature information to the rule $PP \rightarrow PNP$ so that, in combination with appropriate lexical entries, it will generate

but not

[pp *for him*]

pp *for them* I

pp *for he*]

[pp *for the y*)

Write your rule in a form suitable to be added to the grammar we are developing.

5.2.4 How the Rules Fit Together

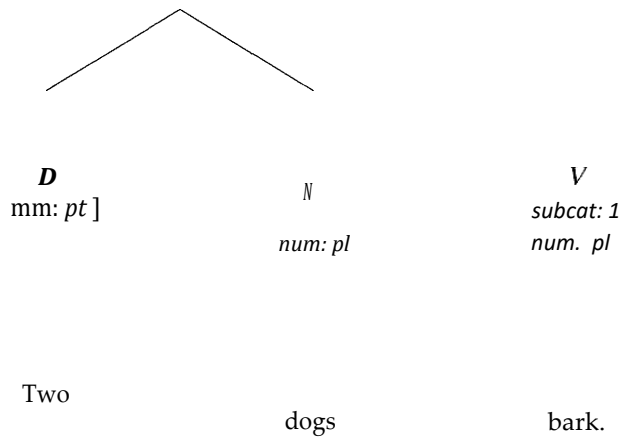
Unification-based rules work equally well when applied bottom-up, top-down, or in any other order. To work out by hand how the rules generate a particular sentence, it is probably easiest to proceed bottom-up.

Consider the sentence *Two dogs bark*. Does our grammar generate it? To find out, first look at the lexical entries for the three words, and fill in the part of the tree that they supply:

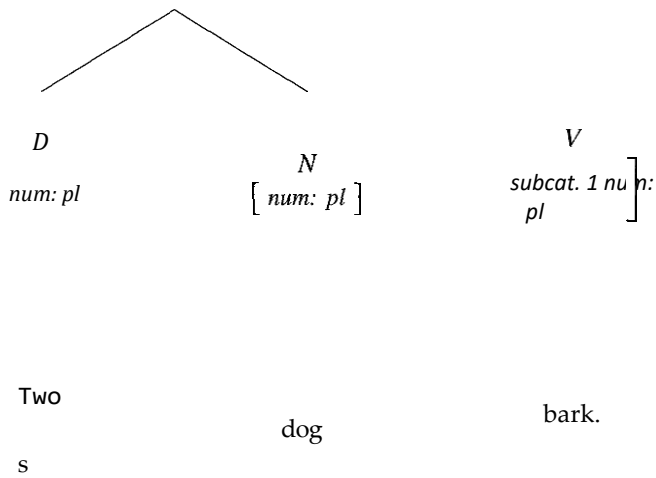
<i>D</i>	<i>N</i>	<i>V</i>
<i>num. pl</i>	[<i>num. pl</i>]	<i>subcat: 1</i>
		<i>mm 1</i>
Two	dog	bark.
s		

Next, group *D* and *N* together into an *NP*. Note that the *NP* → *D N* rule requires *NP*, *D*, and *N* to have the same *mm* feature. So far, no problem:

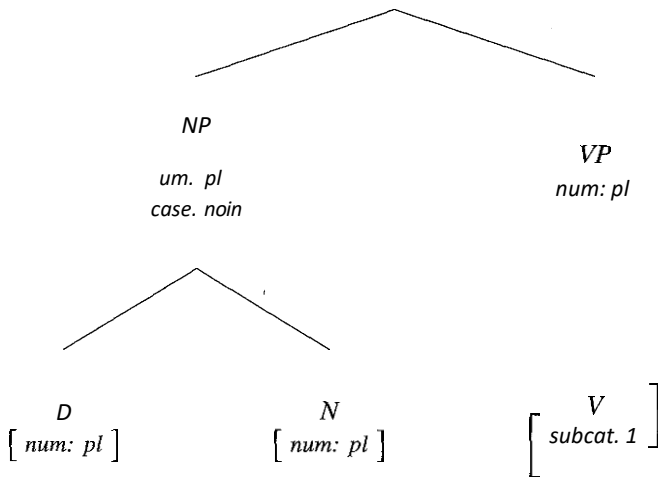
NP
num. pl]



Now we need a VP. The verb has *subcat. 1*. Only one of the two VP rules matches a verb with this feature, namely the rule $VP \rightarrow V$, which percolates the mm feature up



Finally, if *NP VP* requires the NP and VP to agree in number (which they do), and assigns *case: nom* to the NP (which has no visible effect because nouns are not marked for case). Here is the complete structure:



Two

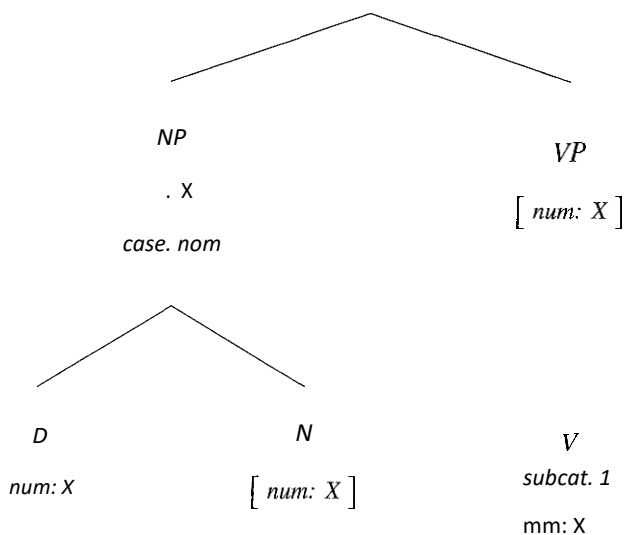
Voila—the

rules generate the sentence.

dogs

bark.

It makes equal sense to work top-down, starting with $S \rightarrow NP VP$, except that the process involves more suspense because there are lots of variables that don't get instantiated until the last moment. For example, applying the rules $S \rightarrow NP VP$, then $NP \rightarrow D N$, and then $VP \rightarrow V$, you get



and when you get to the lexical entries, X finally gets instantiated as *eg* or *pl*.

This illustrates a key advantage of unification—based grammar. You don't have to know the values of the variables in order to manipulate them. As long as the right variables are made equal to each other, and the variables eventually get instantiated, everything comes out correct.

Exercise 5.2.4.1

By working bottom-up, determine whether the grammar generates each of the following sentences. Show your steps. If the sentence turns out not to be generated, show precisely where the unification fails.

1. *The dogs scare him.*
2. *It barks him.*
3. *The cats bark.*

Exercise 5.2.4.2

By working top-down from $S \rightarrow NP VP$, show why the grammar does not generate each of the following sentences. That is, show your steps, and point out the feature conflict when it occurs.

1. *It scares he.*
2. *It scare him.*

5.3 FORMAL PROPERTIES OF FEATURE STRUCTURES

5.3.1 Features and Values

A feature structure is a set of FEATURES (ATTRIBuTEs) and **v**ALUES. It contains at most

contains the value *b* for the feature *n*, the value *d* for the feature *c*, and no value for the feature *e*. But

$$\begin{bmatrix} a. \ b \\ a: \ c \end{bmatrix} \qquad \text{(wrong!)}$$

is not a feature structure, because it does not give a unique value for *a*.

A feature is simply a name; no more, no less. A value can be either an atomic symbol (like a Prolog atom or number), or another feature structure. Some examples:

$$\begin{bmatrix} a. \ b \\ c: \begin{bmatrix} d. \ e \\ fg \\ h: \begin{bmatrix} i: \ j \\ k: \ l \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} semantics. \begin{bmatrix} pred: \textit{chases} \\ tense. \textit{present} \end{bmatrix} \\ syntax: \begin{bmatrix} person. \ 3 \\ number. \ sg \end{bmatrix} \end{bmatrix}$$

Nested feature structures like these allow features to be grouped together. For example, it is often convenient to group all the features that percolate into a structure called *ngr(eement)*.' Then instead of

Pronoun

$$\begin{bmatrix} pers. \ 3 \\ num: \ eg \\ case: . \ acc \end{bmatrix}$$

him

we would write:

Pronoun

$$\begin{bmatrix} \bullet 8r: \begin{bmatrix} yers. \ 3 \\ num: \ *8 \\ case: \ acc \end{bmatrix} \end{bmatrix}$$

→

him

(The pronoun will also have other features that are not in the *agr* group; we just haven't seen them yet.) Then—here's the simplification—it becomes possible to percolate *pers*, *sxm*, and *case* all at once by just percolating *agr*, like this:

<i>NP</i>	<i>Pronoun</i>
<i>agr: X]</i>	<i>[agr.-]</i>

Here is a full description of feature structure unification as we now know it:

'It is not entirely clear whether *case* belongs in the *agr* group. For now, it's convenient to put it there.

In English, case percolates but is not involved in agreement.

- To unify two feature structures, unify the values of all the features.
- If a feature occurs in one structure and not in the other, simply put it into the resulting unified structure.
- If a feature occurs in both structures, unify its values:
 - To unify values that are atomic symbols, check that they are equal; otherwise the unification fails.
 - To unify a variable with anything else, simply make it equal to that thing.
 - To unify values that are feature structures, apply this whole process recursively.

Here are a couple of examples:

$$\begin{array}{l} a: b \\ c: X \end{array} \quad \text{and} \quad \left[\begin{array}{l} \left[\begin{array}{l} d: e \\ f: g \end{array} \right] \\ h: i \end{array} \right] \quad \text{unify to give} \quad \left[\begin{array}{l} a: b \\ c: \left[\begin{array}{l} d: e \\ f: g \end{array} \right] \end{array} \right]$$

$$\begin{array}{l} a: p \\ c: X \end{array} \quad \text{and} \quad \left[\begin{array}{l} a: Y \\ b \\ c! \quad d Y \\ e! f \end{array} \right] \quad \text{unify to give} \quad \left[\begin{array}{l} a: p \\ b \quad d: p \\ e: f \end{array} \right]$$

Without changing the computational power of the formalism, we can in fact allow values to be Prolog terms of any type. This is so because any Prolog term can be translated into a feature structure. For example, $f(a, b)$ could become

$$\left[\begin{array}{l} \text{functor. } f \\ \text{arg1. } a \\ \text{arg2. } b \end{array} \right]$$

and the list $[a, b, c]$ could be rendered as:

$$\left[\begin{array}{l} \text{first: } a \\ \text{rest: } \left[\begin{array}{l} \text{first: } b \\ \text{rest: } \left[\begin{array}{l} \text{first: } c \\ \text{rest: } \text{nil} \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right]$$

So we will frequently use Prolog terms as a substitute for feature structures, wherever this is more convenient. Naturally, we presume that Prolog terms unify by ordinary Prolog unification.

Exercise 5.3.1.1

Unify each of the following pairs of feature structures, or indicate why the unification fails:

$$a [b. \quad \uparrow$$

3. $a: [b, c]$ and $a: [b, X]$
 $d: e$ $d: X$

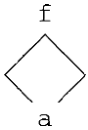
4. $\left[\begin{array}{l} \text{syntax:} \\ \text{category!} \text{ noun} \\ \text{number: plural} \end{array} \right]$ and $\begin{array}{l} \text{syntax. number: } X \\ \text{semantics: number: } X \end{array}$

$\text{semantics. pred. dog}$

5.3.2 Re-entrancy

Feature structures are RE—ENTRANT. Fortunately, this is a property we are familiar with from Prolog. What it means is that if two features have the same value, their values are *the same object*, not merely two objects that look alike.

To take a Prolog example, unify $f(X, X)$ with $I(a(b), \_)$. The result is $f(a(b), a(b))$. But the important thing is that the result doesn't contain two $a(b)$'s; it contains two pointers to the *same* $a(b)$. That is, its tree structure is something like



b

rather than:

$a \quad a$

$b \quad b$

Feature structures work the same way. If you unify

$p: X$ and $p: [a, b]$
 $q: X$

then you get a structure where p and q have the same value, not just two identical-looking values. We will often write this as

$$\left[\begin{array}{l} p. \ [\ a:b \] \\ q. \ \ a:b \] \end{array} \right]$$

but if we do so, something important is lost. To make it perfectly clear that the two instances of [n.fi] are really one structure, we can give it an identifying number and write it only once, thus:

$$\left[\begin{array}{l} p. \ \ \ [a: b \] \end{array} \right]$$

Here the first $\left[\begin{array}{l} p. \ \ \ [a: b \] \end{array} \right]$ serves as a label for the structure [o. b]. By writing $\left[\begin{array}{l} p. \ \ \ [a: b \] \end{array} \right]$ a second time, we indicate that the same structure is also the value of another feature.

We will use this notation when it is absolutely necessary to show that a structure is re—entrant. For the most part, however, re-entrancy will be easy to understand from contCxt, especially if you are familiar with Prolog.

Here is a case where the notation with boxed numbers is helpful. Suppose we’re grouping agreement features into a structure called *Dgr* as proposed a couple of sections back, and now we want to construct an *S NP VP* rule such that:

- the NP and VP share all their *agr* features, i.e., agree in person, number, and whatever else is relevant; and
- the NP has the $\bullet$ *gr* feature *case: nom*.

In effect, we want to combine the two rules

$$\begin{array}{ccc} S & & \begin{array}{cc} NP & VP \\ \rightarrow & [\ agr: X \quad [\ agr: X \] \end{array} \end{array}$$

$[agr: [case: nom]]$

into one. The trouble is that if we give the NP a feature written as *agr. X*, as in the first rule, there's no good way to refer to something within *agr*, as is necessary in the second.

Boxed numbers come to the rescue. We can simply write:

$$S \rightarrow \begin{array}{cc} NP & VP \\ [agr: [case: nom]] & [*8*--p] \end{array}$$

This means: The *agr* features of the NP and of the VP are to be unified with each other, becoming a single structure known as σ . In addition, σ must unify with $[case: nom]$. This has the side effect of giving *case: nom* to the VP, which is harmless because verbs are not marked for case (though it does suggest a reason for not putting *case* in the *agr*

Unify the following pairs of feature structures. Use boxed numbers to indicate all re-entrancy. For example,

$$\llcorner \begin{matrix} Z \\ b: X \end{matrix} \quad \text{and} \quad a: [b: c] \quad \text{unify giving} \quad \begin{matrix} a: \boxed{1} \\ b: \boxed{1} \end{matrix} [b: c]$$

$$\begin{array}{l} 1. \left[\begin{matrix} b: X \end{matrix} \right] \quad \text{and} \quad \left[\begin{matrix} a: [p: q] \end{matrix} \right] \\ 2. \left[\begin{matrix} a: X \\ b: y \end{matrix} \right] \quad \text{and} \quad \left[\begin{matrix} a: p: Y \\ b: e: f \end{matrix} \right] \\ d \end{array}$$

5.3.3 Functions, Paths, and Equational Style

A **PATH** is a description of where to find something in a nested feature structure. For example, in the structure

$$\left[\begin{matrix} a: b \\ p: \left[\begin{matrix} c: \left[\begin{matrix} d: e \\ f: g \end{matrix} \right] \end{matrix} \right] \\ q: r \end{matrix} \right]$$

the path $p : c : d$ leads to the value e .

In mathematics, a **PARTIAL FUNCTION** is a function that yields a value for some arguments but not for others. We can view a path as a partial function which, given a feature structure, may or may not return a value. For example, in the structure above, $p : c : d$ has the value e , and q has the value r , but $p : z : y$ has no value.

This suggests a different way of writing unification-based grammar rules. Instead of using variables in feature structures, we can specify that the values of certain paths have to be equal. I call this “equational style.” For example, instead of

we can write

$S \rightarrow$

NP

$pers: P$

$num: X$

$case: nom$

p
p

pers: P

(NP pers) -- VP pers)

(NP mm) -- (VP num

(NP case) -- nom

That is, the NP's *pers* equals the VP's *pers*, and the NP's *mm* equals the VP's *num*, and the NP's *case* equals *com*. This is the notation used by the computer program PATR-II (Shieber 1985).

We are about to develop an extension of Prolog called GULP in which both conventional and equational styles can be used. Here are two ways of writing the above rule in GULP:

```
s --> np(pers:P..num:X..case:nom), vp(pers:P..num:X) .

s --> np(NPfeatures), vp(VPfeatures),
      ( NPfeatures = pers:P,   VPfeatures = pers:P,
        NPfeatures = num:X,   VPfeatures = num:X,
        NPfeatures = case:nom ).
```

In both versions, *np* and *vp* each have a single Prolog term as an argument. In the first version, each of these arguments is a feature structure. In the second, the arguments are variables (*NP features* and *VP features*) which must then be unified with several feature structures; for example, *NP features* is unified, in succession, with *hers. P*, [*sum. X*], and [*case: nom*]. This won't work in ordinary Prolog, of course, but GULP translates notations such as *case: nom* into structures that unify in the desired way.

Crucially, both GULP and PATR—II allow you to use paths. For example, in PATR—II, you could describe

$$NP$$

$$[agr. [case. nom]]$$

by saying (*NP agr case*) — *com*. In GULP, you could say that NP has the feature *agr:case:nom*.

Exercise 5.3.3.1

Identify all the paths in

$$\left[\begin{array}{l} p: \left[\begin{array}{l} a. b \\ c: \left[\begin{array}{l} d. e \\ f: g \end{array} \right] \end{array} \right] \\ q: r \end{array} \right]$$

and give their values.

Exercise 5.3.3.2

Express

$$S \rightarrow \begin{array}{cc} NP & VP \\ agr: [case: nom]] & [agr. 1] \end{array}$$

in PATR-II equational style. (Hint: You will need to mention the path *NP agr case* in one

5.4.1 A Better Syntax for Features

So far we have implemented features in Prolog as arguments to nodes in DCGs. This works well as long as there are only a few features. Clearly, however, a wide-coverage parser for a human language will have many features, perhaps dozens of them. Most rules mention only a few features, but argument positions have to be provided for every feature so that terms will unify properly. It is easy to end up with grammar rules such as

```
s( _, _, _, Tns, _, s(Tree1, Tree2), _, _, _ ) -->
    np( _, _, _, N, Pers, Tree1, _, _ ),
    vp( _, _, _, Tns, N, Pers, Tree2, _, _, _ ).
```

which are hard both to read and to type correctly.

To make features less cumbersome we need to do three things:

- Collect all the features into feature structures. Then each node will have only one argument, a feature structure.
- Develop a convenient notation for writing feature structures in Prolog.
- Somehow get feature structures to unify properly.

The notation for feature structures is not too hard. All that is necessary is to define the infix operators `:` and `..` and write

$$\left[\begin{array}{l} \text{case: } \textit{nom} \\ \text{person: } P \\ \text{Mem: } \textit{pred. bark} \\ \text{arg1: } \textit{fido} \end{array} \right]$$

as `case : nom .. person : P .. Mem : (pred: bark .. arg1 : fido)`. That is, `:` joins a feature to a value, and `..` joins one feature—value pair to the next.² The value of a feature can be any Prolog term, or another feature structure. This is known as GULP notation because it was first used in a program known as Graph Unification Logic Programming (Covington 1989). Fig. 5.2 shows further examples of GULP notation.

The next question is how to get feature structures to unify. There are two possibilities:

- We could write our own unifier that will handle feature structures written in GULP notation; or

¹By using colons, we make the Prolog module system unavailable, but this is only a minor limitation; a different character could easily be used. Some Prologs may require you to use something other than the colon.

In Arity Prolog 4, a blank is required before each left parenthesis within a feature structure.

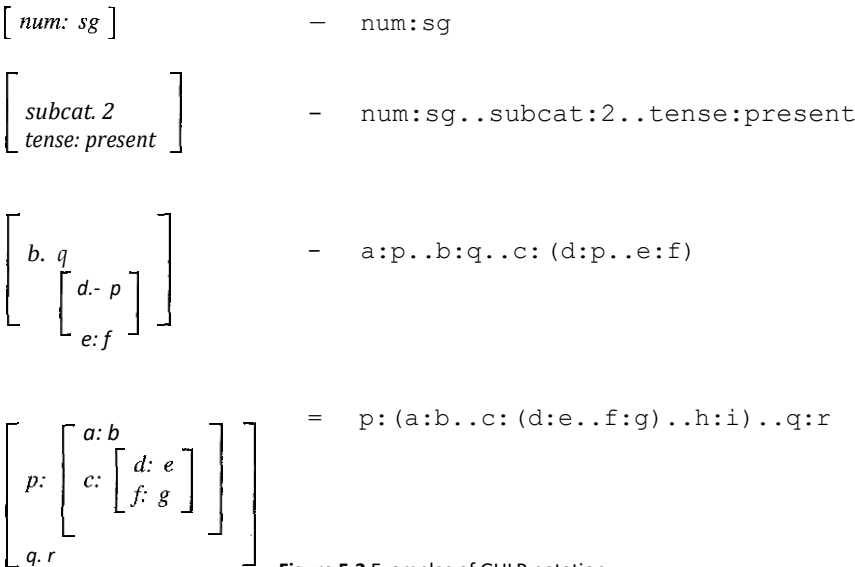


Figure 5.2 Examples of GULP notation.

- We could write a translator that will convert GULP feature structures into something that the ordinary Prolog unifier handles in the desired way.

Johnson and Klein (1986) and others have taken the first approach; here we take the second.

All we really have to do is map names of features onto positions in a list. For example, if *case*, *person*, and *number* are the only features in our grammar, we can represent each feature structure as a list

$[C, P, N]$

where C stands for the case, P stands for the person, and N stands for the number. Then in order to unify

$\left[\begin{matrix} \end{matrix} \right]$ 797 *case. nom person. 3*

and

c

$\left[\begin{array}{l} ase. nom \\ number: eg \end{array} \right]$

we simply unify $l\ nom, 3,)$ with $nom, p, sg)$ and everything comes out right.

So far, so good. Our translator will read a Prolog program, scan it for GULP feature structures, and convert them all into lists with each value in the right position. To keep these lists from being mistaken for ordinary lists in the translated program, we will mark them with the functor $gp/1$; the resulting structure will be called a gp -list. For example:

$\text{case} : C . . \text{number} : N$ translates to $\text{gp} ([C, p, N])$.

This assumes that *case*, *person*, and *number* are the only features in the grammar. If any other features were ever used, there would have to be positions for them in every list.

Crucially, the order of the feature-value pairs does not matter; $\text{case} : C . . \text{number} : N$ translates to exactly the same thing as $\text{number} : N . . \text{case} : C$.

Exercise 5.4.1.1

Write each of the following feature structures in GULP notation.

i. $\text{case} : \text{acc}$

2.
$$\left[\begin{array}{l} \text{case} : \text{nom} \\ \text{person} : 3 \\ \text{num} : \text{pl} \end{array} \right]$$

3.
$$\left[\begin{array}{l} \text{case} : C \\ \text{person} : P \end{array} \right]$$

4.
$$\left[\begin{array}{l} a : b \\ c : \left[\begin{array}{l} d : e \\ h : \left[\begin{array}{l} g \\ k : l \end{array} \right] \end{array} \right] \end{array} \right]$$

5.
$$\left[\begin{array}{l} \text{semantics} : \left[\begin{array}{l} \text{pred} : \text{chases} \\ \text{tense} : \\ \text{present} \end{array} \right] \\ \text{syntax} : \left[\begin{array}{l} \text{person} : 3 \\ \text{num} : \text{eg} \end{array} \right] \end{array} \right]$$

Exercise 5.4.1.2

Lists are not necessarily the best internal representations for feature structures; they are just the simplest to build. Suggest some alternative data structures that could be used instead, and indicate how they would work.

5.4.2 Translating a Single Feature Structure

What we want to build is a translator program, called Mini-GULP, which will accept Prolog programs (including DCGs) that use GULP notation, and will translate the feature structures from GULP notation into gp-lists while leaving the rest of the program unaltered.

The translator program will begin with the op declarations

```
- op(600,xfy,':').
- op(601,xfy,'..').
```

so that as soon as it is loaded, all subsequent Prolog code, including input accepted through `reed /1`, can use GULP notation.

At the beginning of the program to be translated, we will ask the user to supply a set of *schemas* that map features onto positions in a list. Some examples:

```
g_schema(case:X, [X,_,_]).
g_schema(person:X, [_,X,_]).
g_schema(number:X, [_,_,X]).
```

Creation of schemas could easily be automated, and in the full GULP system, it is. Here, however, we're trying to keep things simple.

Each schema then provides the translation of one feature—value pair, except that `'g_/1` is left out. For example:

```
?- g_schema(number:plural,What).
What    [_,_,plural]
```

To translate a series of feature-value pairs, such as

```
case:nom..person:3..number:plural
```

all we need to do is translate the individual feature—value pairs, then unify the

translations with each other, and add 'gp/1 at the beginning. That is, we need to obtain the translations

```
(_,_,plural]
```

and unify them, thereby obtaining $\text{nom}, 3, \text{plural}]$, and then add 'g_' giving $\text{gp}(\text{nom}, 3, \text{plural})$. So far, then, the translator needs two clauses:

```
% g_translate(+FeatureStructure,-g_(List)) (FIRST VERSION)
% Translates FeatureStructure to internal representation g_(List).

% Case 1: A single feature-value pair
g_translate(F:V,g_(List)) -
```

```

g_translate(First..Rest,g_(List))
  g_translate(First,g_(List)),
  g_translate(Rest,g_(List)).

```

This produces the translations that we want, such as:

```

?- g_translate(case:nom..number:plural,What).
What = g_([nom,_,plural])

```

The alert reader will have noticed that all functors that have special meaning for Mini-GULP begin with ‘gp’ to avoid conflict with anything in the user’s program.

Exercise 5.4.2.1

Suppose a Prolog neophyte says, “I don’t understand how `g_translate` adds ‘g_’ to the translation. There isn’t a step in it to do that.” How would you respond?

Exercise 5.4.2.2

Get `g_translate` working. (Remember that the `op` declarations go in the translator program.) Then supply schemas for a grammar containing the features *case*, *number*, *person*, *sem*, *pred*, and *arg1* and give the translations produced by `g_translate` for:

```

case:nom
case:acc..person:2
person:2..case:acc

number:singular..sem:(pred:chases..arg1:fido)..person:3

```

What is unsatisfactory about the last of these?

Exercise 5.4.2.3

Modify `g_translate` so that if the input contains a feature for which there is no schema, an error message will be produced. (This modification is very helpful in doing the subsequent exercises, and in working with Mini-GULP generally. Everyone misspells a feature or leaves out a schema sooner or later.)

5.4.3 Translating Terms of All Types

So far, our translator has two serious limitations:

- It won't translate general Prolog terms, only feature structures.
- It won't accept a feature structure within a feature structure.

For Mini—GULP to be useful, both of these problems have to be corrected. Let's tackle the first one first.

The user's program is nothing more than a series of terms, most of which have the principal functor : – or – >'. What we want to do is search through each term, translating all the feature structures wherever they occur, but otherwise leaving things unchanged. From this perspective, we need to look for *four* kinds of terms, not just two:

- Feature-value pairs. We already know how to handle these, except that we need to *translate* the value, rather than just inserting it into the list unchanged. That will take care of feature structures within feature structures.
- Sequences of feature-value pairs joined by . . '. Just translate all the feature-value pairs and merge the results, as we're already doing.
- Structures. Break the structure up into functor and arguments, recursively translate all the arguments, then reassemble the result.
- Variables and atomic terms. Leave these unchanged. Actually, we check for these *first*, because a variable would match anything, and also because these involve the least work.

Note that this is fully recursive. A feature structure can occur inside any other kind of term, and any other kind of term can occur within a feature structure. Here are the clauses to implement it:

```
% g_translate(+FeatureStructure,-g_(List)) (SECOND VERSION)
% Translates FeatureStructure to internal representation g_(List).

% Case 1: A variable or atomic term

g_translate(X,X)
    (var(X) ; atomic(X)), '.
```

```

% Case 2: A single feature-value pair

g_translate(F:V,g_(List))

    g_translate(V,TranslatedV),
    g_schema(F:TranslatedV,List).

% Case 3: A series of feature-value pairs

g_translate(First..Rest,g_(List))

    g_translate(First,g_(List)),
    g_translate(Rest,g_(List)).

% Case 4: A structure

```

```

g_translate_aux(Args, NewArgs), % translate all args
Result =.. [Functor/NewArgs].

```

Here get rank 1 at epoux / 2 translates, in succession, all the elements in a list, like this:

```

g_translate_aux([TlTerms], [NewTlNewTerms])
  g_translate(T, NewT),
  g_translate_aux(Terms, NewTerms) .

g_translate_aux([], []).

```

Exercise 5.4.3.1

Why doesn't g_C runs 1 at e need another clause to deal with lists?

Exercise 5.4.3.2

Get g_Lrunslat e working and use it to translate the same feature structures as in Exercise 5.4.2.2.

5.4.4 Translating While Consulting

We want the translator to accept programs like that shown in Fig. 5.3. We will write a procedure called gpc onsul t to load such programs into memory, translating GULP notation into g -lists as it does so. The gpc onsu1 procedure will read terms from a file one by one and process them, thus:

- If the term is end o f f i l e, stop. (Recall that this is what read returns when it hits end of file.)
- If the term is a gps cbema, assert it into memory.
- If the term is a grammar rule (with principal functor $\rightarrow$), translate it, then pass it through the DCG rule translator and assert the result into memory.
- If the term is anything else, translate it and then assert it.

The top level of this processing is easy:

```

% g_consult(+File)

% Reads clauses from File, translating as appropriate.

```

```
g_consult(File)
  see(File),
  repeat,

  read(Term),

  g_consult_aux(Term),    % handle it appropriately
```

```

% DCG parser for the grammar in Section 5.2.
% To be processed by Mini-GULP.
% Demonstrates number agreement, case assignment,
% and verb subcategorization.

g_schema(case:X, [X,_,_]).
g_schema(num:X,  [_ ,X,_]).
g_schema(subcat:X,[_ ,_,X]).

pronoun(case:nom..num:sg) --> [he].
pronoun(case:acc..num:sg) --> [him].
pronoun(num:sg)           --> [it].
pronoun(case:nom..num:pl) --> [they].
pronoun(case:acc..num:pl) --> [them].

n(num:sg) --> [dog]; [cat].
n(num:pl) --> [dogs]; [cats].

d(_)      --> [the].
d(num:sg) --> [a].
d(num:pl) --> [two].

v(num:sg..subcat:1) --> [barks].
v(num:pl..subcat:1) --> [bark].
v(num:sg..subcat:2) --> [scareJ].
v(num:pl..subcat:2) --> [scare].

np(num:N) --> d(num:N), n(num:N).

```

```

np(num:N..case:C) —> pronoun(num:N..case:C).

vp(num:N) --> v(subcat:1..num:N).
vp(num:N) —> v(subcat:2..num:N), np(case:acc).

s --> np(case:nom..num:N), vp(num:N).

```

Figure 5.3 Example of a program to be input to Mini-GULP.

```

Term == end_of file,

seen.

g_consult(_) :-      % if something went wrong in previous clause
    seen,
    write('g_consult failed.').

```

```

g_consult_aux(end_of_file) :- ' .

g_consult_aux(g_schema(X,Y)) -

    assertz(g_schema(X,Y)) .

g_consult_aux((X-->Y))

    g_translate((X-->Y),Rule),
    expand_term(Rule,NewRule),      % DCG translator
    assertz(NewRule) .

g_consult_aux(Term) -
    g_translate(Term,TranslatedTerm),
    assertz(TranslatedTerm) .

```

Now Mini-GULP is ready for use. The normal way to use it is as follows:

1. Get into Prolog.
2. Type `? - consult1C(//ennme).` to load Mini-GULP into memory. This executes the op declarations so that GULP syntax becomes legal.
3. Type `? - gpc_consul L(//ename).` to translate and load your program.
4. Type whatever queries your program expects. For example, `? - s([two, cat s, bark], )` would be appropriate for the program in Figure 5.3.

Exercise 5.4.4.1

What happens if you `gpc_consul1L` the same file twice in succession?

Exercise 5.4.4.2

Get `gpc_consul1C` working and use it to translate and run the program in Figure 5.3. (Your translator program should now contain the op declarations, followed by `gpc_consul1at`, `g_consul L`, and `g_consult1t_aux`.)

Exercise 5.4.4.3

Why is the second clause of `g_consul L` necessary?

5.4.5 Output of Feature Structures

There's still one thing missing: a neat way to output feature structures. This is important because many of our parsers will report their results by building a feature structure.

'In ALS Prolog, `expand term/2` is called builtin `:deg.expand/2`, and you have to consult `(dcgs)` to make it available.

Sticking with the program in Figure 5.3, let's take a simple example. If you want to find out the features of [*him*], you can type

```
?- np(Features, [him], []).
```

but you'll merely get

```
Features = g_([acc,sg])
```

which doesn't give the names of the features. In a large grammar with dozens of features, output in this format would be almost useless.

One way to get the computer to report names with the features is to use `gpschema` like this:

```
?- np(g_(Features), [him], []), g_schema(FV, Features).  
FV = case:acc  
FV = num:sg  
no
```

Notice that '`g_`' has been added in the first argument of `np`, so that `Features` is now just a list. Now we get the names and values of all the features, one at a time, as alternative solutions to the `gpschema` subgoal.

That's still rather clumsy, but we can do better. The built-in predicate `setof` will gather all these alternative solutions into a list, like this:

```
?- np(g_(Features), [him], []), setof(FV, g_schema(FV, Features), L).  
L = [case:acc, num:sg]
```

That's *almost* what we need. The problem is that, in most feature structures in most real grammars, most of the features are uninstantiated, so in a larger grammar you'd get something like

```
[person:_001,case:acc,gender:_002,tense:_003,num:pl,sem:_004]
```

which is hardly ideal.

Let's develop a predicate `gym_i C e / 1` that outputs feature structures in readable form. Basically, here's what `gwr i C e` will do:

- Output a `gp—list` by converting it into a series of feature-value pairs (using `g_wr i C e` recursively to write the values, and skipping the ones whose values are uninstantiated).
- Output anything else by calling `or i e / 1`.

The alert reader will notice that this is only partly recursive: `gwr i L e` can handle

```

% g_write(+g_(List))

% Produces legible output of a feature structure in internal form.
% Assumes all necessary schemas are present.
% Imperfect; limitations are noted in text.

g_write(g_(Features)) -

    write(' (',
    setof(FV,g_schema(FV,Features),FVList),
    g_write_aux(FVList),

    write(') ').

g_write(X) -
    write(X).

g_write_aux([])

g_write_aux([_:VIRest])
    var(V),

    g_write_aux(Rest).

g_write_aux([F:VlRest]) -

    write(F) ,
    write(':'),
    g_write(V),
    write('..'),
    g_write_aux(Rest).

g_write_aux(X) -
    write(X).

```

This is good enough to output most feature structures, but (apart from not being completely recursive) it has a couple of flaws: it writes an extra `..'` after the last feature—value pair, and it fails to correctly write a variable or an empty list (and thus is not a perfect substitute for `writeln`). But at least we can now do things like this:

```
?- np(Features,[she],[]), g_write(Features).  
(person:3..number:singular..)
```

and thus we will be able to look at the feature structures produced by more complex grammars.

Exercise 5.4.5.1

Get `gprule` working. A good way to test it is to write a program that consists of a set of schemas such as:

```
g_schema(person:X, [X,_,_]) .
g_schema(number:X, [_,X,_]) .
g_schema(sem:X, [_,_,X]) .
```

followed by some clauses such as:

```
test1 :- g_write(person:1) .
test2 :- g_write(person:3..number:plural) .
test3 :- g_write(sem:(person:3)..number:singular) .
```

Then load the program through `gpc` in order to turn all the feature structures into `g_lists`, and see if `gprule` translates them back correctly when `test1`, `test2`, etc., are executed.

Exercise 5.4.5.2

Modify `gwrite` so that `?- gwrite([]).` will output `[]` and `?- gwrite(X).` will output a representation of an uninstantiated variable (something like `p0_01`, or whatever your Prolog normally produces).

Exercise 5.4.5.3

Modify `gwrite` so that there will not be an extra `..` after the last feature-value pair. Note that the last feature-value pair to be printed is not necessarily the last one in the `g_list`, because features with uninstantiated values are skipped.

5.5 UBG IN THEORY AND PRACTICE

5.5.1 A More Complex Grammar

It's time to look at, and implement, a more elaborate unification-based grammar. Here is a grammar based on the four PS rules:

<i>NP</i>		<i>Fido, Felix, he, him, the y, them</i>
<i>V</i>	$\rightarrow$	<i>chase, chases, sleep, sleeps</i>
<i>VP</i>		<i>V (NP)</i>
<i>S</i>	$\rightarrow$	<i>NP VP</i>
	$\rightarrow$	

To these we will add features to handle the following things:

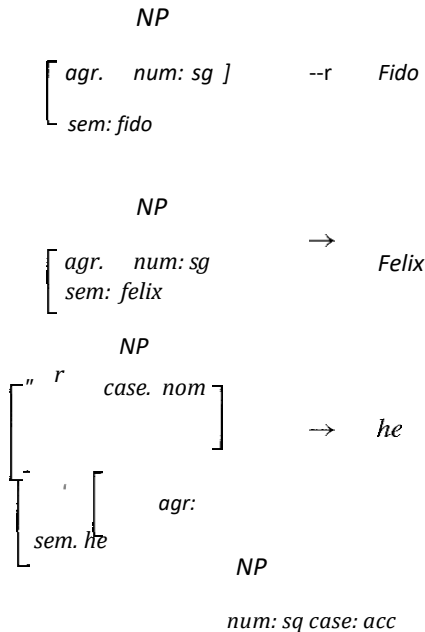
- Case assignment (*he chases me*, not **him chases I*),
- Subject—verb agreement (*Fido sleeps*, not *Fido sleep*),

primitive for practical use, but adequate to illustrate some techniques).

All together, the features that we use will be *agr*, *sem*, *num*, *case*, *pred*, *arg1*, *arg2*, and *subcat*. So the parser has to start with a set of schemas:

```
g_schema (agr:X,      [X,_,_,_,_,_,_,_] ) .
g_schema (sem:X,      [_,X,_,_,_,_,_,_] ) .
g_schema (num:X,      [_,_,X,_,_,_,_,_] ) .
g_schema (case:X,     [_,_,_,X,_,_,_,_] ) .
g_schema (pred:X,     [_,_,_,_,X,_,_,_] ) .
g_schema (arg1:X,     [_,_,_,_,_,X,_,_] ) .
g_schema (arg2:X,     [_,_,_,_,_,_,X,_] ) .
g_schema (subcat:X,   [_,_,_,_,_,_,_,X] ) .
```

Now for the grammar itself. Let's look first at the lexical entries for the NPs. These endow each NP with a semantic representation of sorts, plus a group of *agr* features (number, and case if case is marked).



→ <i>

NP
[agr: [mm: pl
 case: nom]] --+ they
sem. the y

NP
[agr: [num. pl
 case: acc]] --r them
sem. them

They go into DCG with GULP
straightforwardly:

```
np(agr:(num:sg)..sem:fido)
np(agr:(num:sg)..sem:felix)
np(agr:(num:sg..case:nom)..sem:he)
np(agr:(num:sg..case:acc)..sem:him)
np(agr:(num:pl..case:nom)..sem:they)
np(agr:(num:pl..case:acc)..sem:them)
```

subcat: 2

a
g
r
.

n
u
m

.
p
l
s
e
m
.
c
h
a
s
e

As before, we have two subcategories of
verbs:

$$\left[\begin{array}{l} \textit{subcat: 1} \\ \textit{agr: num.sg} \\ \textit{sem: sleeps} \end{array} \right]$$

$$\left[\begin{array}{l} \textit{subcat: 1} \\ \textit{agr: [num: pl} \\ \textit{sem: sleep} \end{array} \right]$$

$$\left[\begin{array}{l} \textit{subcat. 2} \\ \textit{agr: [num: sg} \\ \textit{sem: chases} \end{array} \right]$$

These, too, go into GULP
straightforwardly:

```

--> [fido].
--> [felix].
-> [he].
--> [him].
-> [they].
--> [them].

→ sleep

→ chases

```

```

sleeps → chase

```

```

v(subcat:1..agr:(num:sg)..sem:sleeps)  —> [sleeps].
v(subcat:1..agr:(num:pl)..sem:sleep)    -> [sleep].
v(subcat:2..agr:(num:sg)..sem:chases)   --> [chases].
v(subcat:2..agr:(num:pl)..sem:chase)     -> [chase].

```

Notice that our approach to semantics here is extremely naive—we’re just writing each word itself in place of its semantic representation. This is enough to show that we can get the symbols to come out in the right places; we’ll explore semantics in depth in Chapters 7 and 8.

The VP rules must enforce subcategorization and, if there is an object, assign accusative case to it. Here’s what they look like, in UBG and in GULP:

$$\begin{array}{ccc}
 \begin{array}{c} \text{VP} \\ \left[\begin{array}{l} \text{agr: } X \\ \text{sem: } \left[\begin{array}{l} \text{pred: } P \\ \text{arg2: } A2 \end{array} \right] \end{array} \right] \end{array} & \rightarrow & \begin{array}{cc} \begin{array}{c} V \\ \left[\begin{array}{l} \text{subcat: } 2 \\ \text{agr: } X \\ \text{sem: } P \end{array} \right] \end{array} & \begin{array}{c} NP \\ \left[\begin{array}{l} \text{agr: } [\text{case: } \text{acc} \\ \text{sem: } A2 \end{array} \right] \end{array} \end{array}
 \end{array}$$

vplagr:X..sem:(pred:P) -->

v(subcat:1..agr:X..sem:P).

vp(agr:X..sem:(pred:P..arg2:A2)) -->

v(subcat:2..agr:X..sem:P),

np(agr:(case:acc)..sem:A2).

Notice what these rules do. Besides enforcing subcategorization, percolating *agr*, and assigning case to the direct object, they also build a semantic representation. The semantics of the verb (*P*) and of the noun *A2*) get combined into a structure representing the semantics of the VP, such as

$$\left[\begin{array}{l} \text{pred. chases} \\ \text{arg2: fido} \end{array} \right]$$

if the VP is *chases Fido*. As you might guess, the S rule is going to add an *arg1*, so that *Felix chases Fido* will come out as:

$$\left[\begin{array}{l} \text{pred. chase} \\ \text{arg1. felix} \\ \text{arg2: fido} \end{array} \right]$$

The S rule itself takes the form

$$\begin{array}{ccc}
 \begin{array}{c} S \\ \left[\begin{array}{l} \text{sem: } 1 \\ \text{arg1: } A1 \end{array} \right] \end{array} & \begin{array}{c} NP \\ \left[\begin{array}{l} \text{agr: } \\ \text{sem: } A1 \end{array} \right] \end{array} & \begin{array}{c} VP \\ \left[\begin{array}{l} \text{agr: } \boxed{2} \\ \text{sem: } \boxed{1} \end{array} \right] \end{array}
 \end{array}$$

Here the S has a single feature, *sem*, which is the same as the *fern* feature of the VP except that it also has to unify with *arg1: Al* (which is how the semantics of the subject gets into it). NP and VP share all their *agr* features and are required to contain *case: com*. To express this rule in GULP, we have to use equational style:

```
s(Sfeatures) --> np(NPfeatures), vp(VPfeatures),
{ Sfeatures = sem:(arg1:Al), NPfeatures = sem:Al,
  Sfeatures = sem:S,          VPfeatures = sem:S,
  NPfeatures = agr:X,          VPfeatures = agr:X,
  NPfeatures = agr:case:nom    }.
```

It's also possible to use a style that is only partly equational, like this:

```
s(sem:S) —> np(agr:X..sem:A1), vp(agr:X..sem:S),
  ( S - arg1:A1,
    X = case:nom ).
```

Here the second line means, “In addition to the value that *s* already has, *S* must be unified with *arg1*:A1,” and likewise for *X* and *case*:nom.

Notice that, since the *sem* of the *S* and of the *VP* are unified with each other, *arg1* gets added to the *sem* of the *VP* as well as of the *S*. This is harmless and, in fact, correctly reflects the fact that the meaning of the verb is incomplete until the meaning of the subject is added.

To parse a sentence, issue a query such as:⁴

```
°- s(Features, [felix,chases,fido],[], g_write(Features).
(sem:(arg1:felix..arg2:fido..pred:chases))
```

All you're doing here is invoking a DCG parser in the usual way and telling it to parse [*I* *ido*, *chases*, *Tel ix*] and end up with *.* The argument of *s* is *FeaLur es*, a GULP feature structure which is then printed out by *g_wri C e*. In this grammar, the features of *S* contain a primitive semantic representation of the sentence.

You can equally well parse any other constituent, for example a *VP*, like this:

```
?- vp(Features, chases,fido],[], g_write(Features).
(agr:(num:sg)..sem:(arg2:fido..pred:chases))
```

This highlights an important fact:

Unification-based grammar is not sentence based.

In UBG, the sentence is just one of many constituents that can be described and parsed. This contrasts sharply with transformational grammar, in which many transformations apply only to the whole S, and the grammar does not correctly generate NPs, VPs, etc., unless they are embedded in their proper places in sentences.

Exercise 5.5.1.1

Draw trees (with features) for the following sentences generated by this grammar:

1. Fido chases him.

'The output as shown here assumes that the behavior of g write has been cleaned up as suggested in the exercises. Otherwise there will be a redundant at the end.

In doing this by hand, it is probably best to work bottom-up. First write down the words and apply the lexical entries. Then group the words into phrases, performing appropriate unifications as you go.

Exercise 5.5.1.2

Get this grammar working, as a DCG parser, on your computer. Parse the three sentences from the previous exercise, and give the features of the S.

Exercise 5.5.1.3

Modify this grammar so that every node has another feature, *tree*, whose value is a representation of the parse tree below it. This will work very much like the tree-building parser in Chapter 3. Parse the same three sentences again and show the value of *tree* for S.

Exercise 5.5.1.4

Construct and implement a UBG to parse sentences such as

Who dix:1 Max say thought Fido barked?

(as in Section 3.4.5), constructing a representation of the tree in which *orio* is associated with (or moved into) the position of the missing NP.

5.5.2 Context-Free Backbones and Subcategorization Lists

So far, every UBG that we've worked with has had a CONTEXT-FREE BACKBONE—that is, if you strip away all the features, you get context-free PS rules.

Notice that the node labels themselves can be treated as features. Instead of

NP

case. nom

we can write

[*case: nom*
827

So far so good; the grammar still has a context-free backbone *if* every node in every rule has a *cat* feature whose value contains no variables. (Normally the value of *cat* is an atomic symbol such as *up*, but Jackendoff (1977) and Gazdar et al. (1985) explore what can be accomplished by using feature structures there.)

If we let *car* be a variable, we pass into interesting territory. For one thing, we can handle verb subcategorization by giving each verb a feature which is a *list* of complement categories. Consider the rules:

$$\left[\begin{array}{l} \text{cat. } vp \\ \text{subcat: } Y \end{array} \right] \rightarrow \left[\begin{array}{l} \text{cat. } vp \\ \text{subcat: } [X \ Y] \end{array} \right] \quad [\text{cat: } X]$$

$$[\text{cat: } s] \rightarrow [\text{cat: } np] \left[\begin{array}{l} \text{cat. } vp \\ \text{subcat: } [] \end{array} \right]$$

The first rule is effectively $VP \rightarrow VP \ X$, where *X* comes from the *subcat* list of the *VP*. It applies recursively, picking off values of *X* one by one until there are none left. Then the second rule, $S \rightarrow NP \ VP$, is allowed to apply.

With these rules we use lexical entries such as the following:

$$\begin{array}{lll} \left[\begin{array}{l} \text{cat. } vp \\ \text{subcat: } [] \end{array} \right] & \text{bark(s)} & \text{bark takes no complement} \\ \\ \left[\begin{array}{l} \text{subcat: } (up) \end{array} \right] & \rightarrow \text{chase(s)} & (\text{chase takes one NP}) \\ \\ \left[\begin{array}{l} \text{subcat: } (up,rip) \end{array} \right] & \text{give(s)} & (\text{give takes two NPs}) \\ \\ \left[\begin{array}{l} \text{cat. } vp \\ \text{subcat: } (ny,s) \end{array} \right] & \text{tell(s)} & (\text{tell takes an NP and an S}) \end{array}$$

The resulting structures are as shown in Fig. 5.4.

This analysis follows a proposal by Shieber (1985:27–32) except that we use Prolog notation for lists, and the subject of the verb is not listed among its complements. Note that a verb here is a *VP*, not a *V*. Note also that the elements of a subcategorization list could be feature structures.

Unfortunately we can't implement this with a top-down DCG parser. The obvious way to approach it is to give all nodes the same label (let's use *z*) and encode the real node labels in the *cat* feature, rendering the first rule thus:

```
z1cat:vp..subcat:Y) --> z(cat:vp..subcat:[XIY]), z(cat:X).
```

The problem is that, parsing top-down, this rule creates an endless loop. In order to parse `z ( cat : vp . . subcaC : [X Y] )`, with *X* and *Y* uninstantiated, this rule simply calls itself. We will return to this point in the next chapter.

Are subcategorization lists a good idea? That depends. To use subcategorization lists is to claim (with Gross 1979) that verbs do not necessarily fall into classes and thus that the complements of each verb must be listed in its lexical entry. Many generative

[*cat: np*]

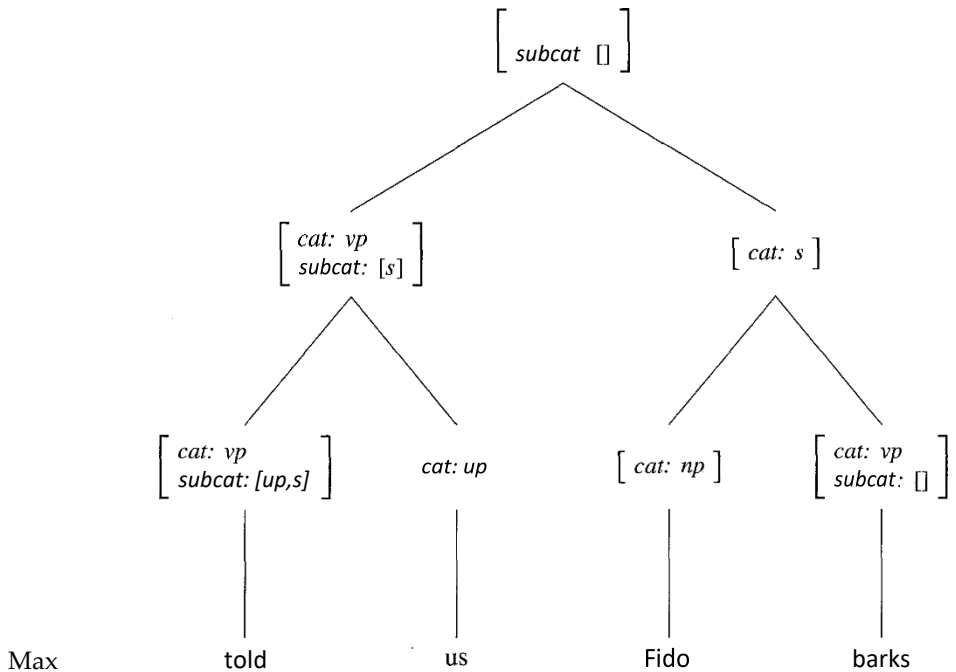


Figure 5.4 Tree showing how subcategorization lists work.

linguists, however, believe (with Gazdar et al. 1985) that verbs fall into a finite, though possibly large, set of classes, and thus that subcategorization with numbers (as pointers to classes) is more appropriate.

Exercise 5.5.2.1

Using the rules developed in this section, draw trees for the sentences:

1. Fido chases Felix.

2. Felix gives Fido a hard time.

Assume any other PS rules that are necessary.

5.5.3 Negative and Disjunctive Features

The alert reader will notice that we have been ignoring the *person* feature of the English verb. There's a good reason. Every regular present tense verb has two forms: with -s in the third person singular, and without -s elsewhere. Thus we get:

	Singular	Plural
1st person	(I) <i>bark</i>	(we) <i>bark</i>
2nd	(you) <i>bark</i>	[you] <i>bark</i>
3rd person	(he, she, it) <i>barks</i>	(they) <i>bark</i>

In UBG as formulated so far, the best we can do is list the form with -s once, and the form without -s three times:

person: '3 → *barks*

V
[*agr.* [*person.* 2]] → *bark*

v
[*agr:* *person:* 1] → *bark*

V
[*agr:* [*num:* *pl*]] → *bark*

It would be more convenient if we could collapse the last three of these into one rule like this:

v

agr: not *Persan.3* --+ *bark*

Here not means ‘*this* feature must ultimately have a value that will not unify with the value shown here.’ This is called a *NzGATivE* feature.

Or consider the forms of the German word for “the” (*der, die, das*, etc.). A striking pattern emerges: the form is *die* whenever:

- the case is nominative or accusative (not genitive or dative), *and*
- the gender is feminine *or* the number is plural (or both).

D

case: nom or acc

die

agr: [num. pl or gen: fern

Here we've moved case outside the *agr* group, and *or* means "the actual value must unify with one of these values, or the other, or both." Features joined with *or* are called DISJUNCTIVE features.

Obviously the Prolog unifier will not handle negative or disjunctive features. Johnson (1991) discusses them lucidly and offers an analysis based on classical logic, leading to a unifier based on a theorem-proving algorithm.

Exercise 5.5.3.1

Using negative and/or disjunctive features, write concise lexical entries for the English copula:

	Singular	Plural
1st person	(I)	are
2nd person	(you)	are
3rd person	(he, she, it)	is

Exercise 5.5.3.2

If negative and disjunctive features were implemented, what should be the result of unifying each of the following pairs of feature structures? Explain your answers and relate them to what you know about logic.

1. [*person*. not 3] and [*person*. 2]
2. [*person*. 1 or 2] and [*person*: 2]
3. [*a*: *b* or *c*] and [*a*: *c* or *d*]
4. [*n*: *b* or *c*] and [*a*: not *b*]

Exercise 5.5.3.3 (project)

Implement a unifier for negative and/or disjunctive feature structures. (See Johnson (1991) for guidance.)

6.1 COMPARING PARSING ALGORITHMS

So far, all our parsers have relied on Prolog's built-in DCG translator as discussed in Chapter 3. In this chapter we will implement a number of parsing algorithms for ourselves. Parts of this chapter are heavily indebted to Pereira and Shieber (1987), who describe several algorithms in more detail.

We will take advantage of the fact that, in Prolog, backtracking is automatic. This makes Prolog ideal for implementing parsers and seeing clearly how they work. Most parsers need to backtrack, but most programming languages make backtracking rather hard to implement. This means that, in ordinary programming languages, the core of any parser is likely to be hidden under a large, cumbersome backtracking mechanism.

All our parsers will use the grammar shown in Figure 6.1, except for certain rules that particular parsing algorithms can't handle.

6.2 TOP-DOWN PARSING

Let's start with top-down recursive-descent parsing. This is the same algorithm used by

NP *D N*
NP *NP Conj NP* (not for top-down parsers)

VP $\rightarrow$ *V NP [PP]*

PP *P NP*

D $\rightarrow$ *B* (not for bottom-up parsers)
D *the, all, every*

P $\rightarrow$ *near*

Conj and

N $\rightarrow$ *dog, dogs, cat, cats, elephant,*
elephants

Figure 6.1 A sample grammar for

V *chase, chases, see, sees, amuse, amuses*

experimenting with parsers.

grammar rules—it translates them directly into executable Prolog clauses. For example,

$\rightarrow$

goes into memory as something like this:

s[L1,L] :- np[L1,L2], vp[L2,L].

What we’re going to build is an INTERPRETER for grammar rules. Instead of executing the rules directly, it will store them as facts in a knowledge base and look them up as needed. Execution will be a little slower, but the parser will be much easier to modify. We’ll represent PS rules as

```
rule(s, [np, vp]).
rule(np, [d, n]).
rule(d, []).
```

and the like. Lexical entries will look like this:

```
word(d, the) .
```

```
word(n, dog) .
```

```
word(n, cat) .
```

```
word(v, chase) .
```

```
word(n, dogs) .
```

```
word(n, cats) .
```

```
word(v, chases) .
```

This format works conveniently with a variety of parsing algorithms; we'll be able to transfer the set of rules with little or no change from one parser to another.

Now recall how top-down parsing works. To parse *The dog barked*, the parser, in effect, says to itself:

- I'm looking for an S.
- To get an S, I need an NP and a VP.
- To get an NP, I need a D and an N.
- To get a D, I can use *the*... Got it.
- To get an N, I can use *dog*.. Got it.
- That completes the NP.
- To get a VP, I need a V.
- To get a V, I can use *barked* ... Got it.
- That completes the VP.
- That completes the S.

To get a feel for what's going on, try drawing the tree as you work through these steps. Figure 6.2 shows the order in which a top-down parser discovers the various parts of the tree.

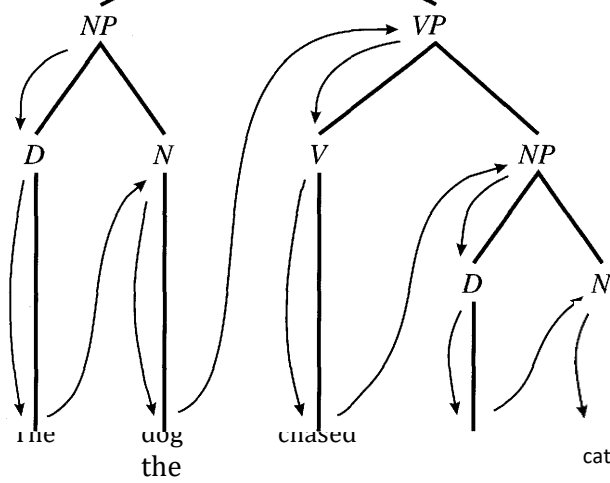


Figure 6.2 Top-down parsing. The parser discovers the nodes of the tree in the order shown by the arrows.

Like the DCG system, our parser will work through the input string, accepting

constituent of type C is:

- If C is an individual word, look it up in the lexical rules and accept it from the input string.
- Otherwise, look in the PS rules to expand C into a list of constituents, and parse those constituents one by one.

Or, putting it into Prolog:

```
% parse(?C, ?S1, ?S)
%   Parse a constituent of category C
%   starting with input string S1 and
%   ending up with input string S.
```

```
parse(C, [WordIS], S)  -
    word(C, Word) .
```

```
parse(C, S1, S)
    rule(C, Cs) ,
    parse_list(Cs, S1, S) .
```

Here `parse_list` is just like `parse` except that it takes a list of constituents and parses all of them:

```
parse_list([C1Cs], S1, S)
    parse(C, S1, S2) ,
    parse_list(Cs, S2, S) .
```

```
parse_list([], S, S) .
```

That's the whole parser. The query to parse a sentence looks like this:

```
?- parse(s, [the, dog, barked], []) .
```

Notice that backtracking, where needed, occurs automatically.

Exercise 6.2.0.1

Get this parser working and equip it with all the grammar rules in Figure 6.1 except for $NP \rightarrow NP \text{ Conj } NP$. Use it to parse sentences such as *All dogs amuse the elephant*.

Exercise 6.2.0.2

In this parser, why is $D \rightarrow 8$ encoded as a PS-rule rather than as a lexical entry?

Exercise 6.2.0.3

By adding arguments to nodes (as in Chapter 3, section 3.4), make the parser enforce subject-verb number agreement and build a representation of the parse tree. (Hint: This is easy; only the rules need to be modified.)

Exercise 6.2.0.4

(Not for courses that skipped Chapter 5.) Combine Mini-GULP with this parser. Account for number agreement of subject and verb, and of determiner and noun, using GULP feature structures.

Exercise 6.2.0.5

Modify the parser so that it always builds a representation of the parse tree, even with the rules in their original form (without arguments). (Hint: Add extra arguments to `per se` and `parse_list`.)

6.3 BOTTOM-UP PARSING**6.3.1 The Shift-Reduce Algorithm**

A familiar limitation of top—down parsers is that they loop on LEFT-RECURSIVE rules of the form $A \rightarrow_r A B$ (“to parse an A , parse an A and then ...”). Yet, as we saw in Chapter 4, such rules occur in natural language; one example is $NP \quad NP \text{ Conj } NP$, where *Conj* is a conjunction such as *and* or *or*.

One way to handle left—recursive rules is to parse BoTTOM—UP. A bottom—up parser accepts words and tries to combine them into constituents, like this:

- Accept a word ... it's *the*.
- *The* is a D.

- Accept another word ... it's *dog*.
- *Dog* is an N.
- D and N together make an NP.
- Accept another word ... it's *barked*.
- *Barked* is a V.
- V by itself makes a VP.
- NP and VP make an S.

Again, try drawing a tree while working through these steps. (During most of the process it will of course be a set of partial trees not yet linked together.) You'll find that you encounter the various parts of the tree in the order shown by the arrows in

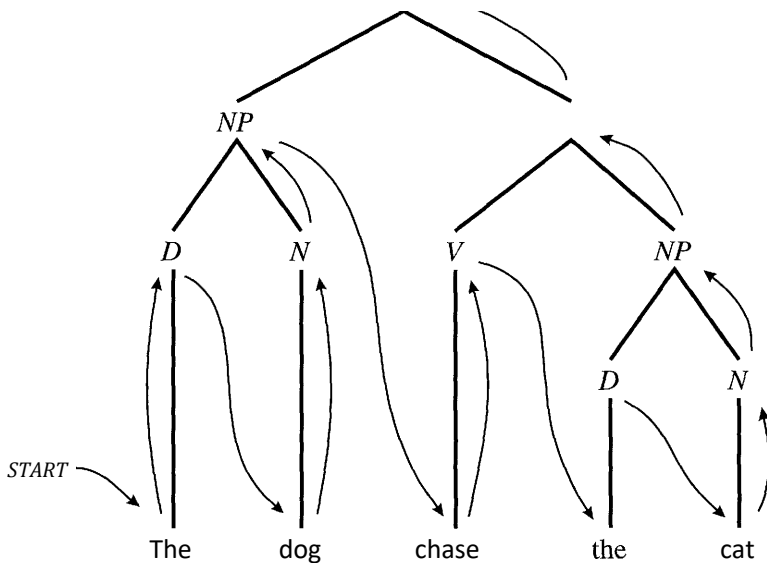


Figure 6.3 Bottom—up parsing. The parser discovers the nodes of the tree in the order shown by the arrows.

Because its actions are triggered only by words actually found, this parser does not loop on left—recursive rules. But it has a different limitation: it cannot handle rules like

$$D \rightarrow \emptyset$$

because it has no way of responding to a null (empty, missing) constituent. It can only respond to what's actually there.

The bottom-up algorithm that I've just sketched is often called **SHIFT-REDUCE** parsing. It consists of two basic operations: to **SHIFT** words onto a stack, and to **REDUCE** (simplify) the contents of the stack. Here's an example:

Step	Action	Stack	Input string
	(Start)		<i>the dog barked</i>
			<i>barked</i>
1	Shift	<i>the</i>	<i>dog barked</i>
2	Reduce	<i>D</i>	<i>dog barked</i>

3	Shift	<i>D dog</i>	<i>barked</i>
4	Reduce	<i>D N</i>	<i>barked</i>
5	Reduce	<i>NP</i>	<i>barkecl</i>
6	Shift	<i>NP barked</i>	
7	Reduce	<i>NP V</i>	
8	Reduce	<i>NP VP</i>	
9	Reduce	<i>S</i>	

So the shift—reduce algorithm is as follows:

1. Shift a word onto the stack.
2. Reduce the stack repeatedly using lexical entries and PS rules, until no further reductions are possible.
3. If there are more words in the input string, go to Step 1. Otherwise stop.

A noteworthy feature of shift-reduce parsing is that it has no expectations. That is, you can give it an input string and say, “What kind of constituent is this?” and get an answer. You do not have to say “Parse this as an S” or “Parse this as an NP.”

Exercise 6.3.1.1

By hand, using the grammar in Figure 6.1, do shift-reduce parsing on the following input strings:

1. *the dogs amuse the elephants*
2. *see the cat near the dog*
3. *the dogs and the cats*
4. *the dogs the dogs the dogs*

What happens in the last of these?

6.3.2 Shift-reduce in Prolog

Efficient shift-reduce parsing in Prolog requires a couple of subtle techniques. The first of these is to build the stack backward. That is, shift words from the beginning of the input string to the *beginning* of the stack, so that they end up in reverse order, thus:

Start:	[]	[the, dog, barked]
Shift:	[the]	[dog, barked]

Reduce:	[d1	[dog,barked]
Shift:	[dog,d]	[barked]
Reduce:	[n,d]	[barked]
Reduce:	[np]	[barked]
Shift:	[barked,np]	[]
Reduce:	[v,np]	[]
Reduce:	[vp,np]	[]
Reduce:	?s%	[]

This is efficient because all the action is at the beginning of each list; there is no need to work all the way along a list to get to the end.

The second subtlety is that the “reduce” step goes a lot faster if the rules, too, are stored backward. For example,

`brule([n,d1X],[np1X]).`

Here `brule` stands for “backward rule.” The first argument, `[n,d1X]`, directly matches the stack to which this rule applies, and `[np1X]` is what the stack becomes after reduction. With the rule written this way, unification does all the work.

So the “reduce” step in the parser is very simple: if there is an applicable rule, use it, and then try to reduce again; otherwise leave the stack unchanged. The rest of the parser is even simpler; the whole thing is shown in Figure 6.4.

Exercise 6.3.2.1

Using the parser in Figure 6.4, what is the effect of each of the following queries? Consider all possible solutions.

```
° parse([the,dog,chases,the,cat],[s])
```

```
?- parse([the,dog,chases,the,cat],[What]).
```

```
?- parse([the,dog,chases,the,cat],What).
```

Exercise 6.3.2.2

Why isn’t there a cut in the first clause of `reduce`?

Exercise 6.3.2.3

Extend the parser in Figure 6.4 to contain all the grammar rules in Figure 6.1 except for $D \rightarrow r B$. Then use it to parse the input strings from Exercise 6.3.1.1.

Exercise 6.3.2.4

Add features to the shift-reduce parser to enforce subject-verb number agreement and build a representation of the parse tree. As before, only the rules need to be modified.

Exercise 6.3.2.5

(Not for courses that skipped Chapter 5.) Combine Mini-GULP with this parser. Account for number agreement of subject and verb, and of determiner and noun, using GULP feature structures.

Exercise 6.3.2.6

What, exactly, does a shift-reduce parser do if it tries to use a rule like $fi \rightarrow + 8$?

6.4 LEFT-CORNER PARSING

6.4.1 The Key Idea

Left-corner parsers are often described as bottom-up, but left — corner parsing is actually a combination of bottom-up and top-down strategies. This technique was popularized

```

% Bottom-up shift-reduce parser

% parse(+S,?Result)
%   parses input string S, where Result
%   is list of categories to which it reduces.

parse(S,Result)
    shift_reduce(S,[],Result).

% shift reduce(+S,+Stack,?Result)
%   parses input string S, where Stack is
%   list of categories parsed so far.

shift_reduce(S,Stack,Result)
    shift(Stack,S,NewStack,S1),      % fails if S = []
    reduce(NewStack,ReducedStack),
    shift_reduce(S1,ReducedStack,Result).

shift_reduce([],Result,Result).

% shift(+Stack,+S,-NewStack,-NewS)
%   shifts first element from S onto Stack.

shift(X,[HIY],[HIX],Y).

% reduce(+Stack,-ReducedStack)
%   repeatedly reduces beginning of Stack
%   to form fewer, larger constituents.

reduce(Stack,ReducedStack) -
    brule(Stack,Stack2),
    reduce(Stack2,ReducedStack).

```



```

reduce(Stack,Stack).

% Phrase structure rules
brule((vp,nplX],[s1X]).
brule([n,d1X],[nplX]).
brule([np,v1X],[vp1X]).
brule([WordIX1,[CatIX)] :- word(Cat,Word).

% Lexicon
word(d,the).

word(n,dog).      word(n,dogs).
word(n,elephant). word(n,elephants).

word(v,chase).    word(v,chases).
word(v,see).      word(v,sees).

% etc.

```

The key idea is to accept a word, figure out what kind of constituent it marks the beginning of, and then parse the rest of that constituent top-down. The tree is thus discovered starting at the lower left corner, as shown in Figure 6.5.

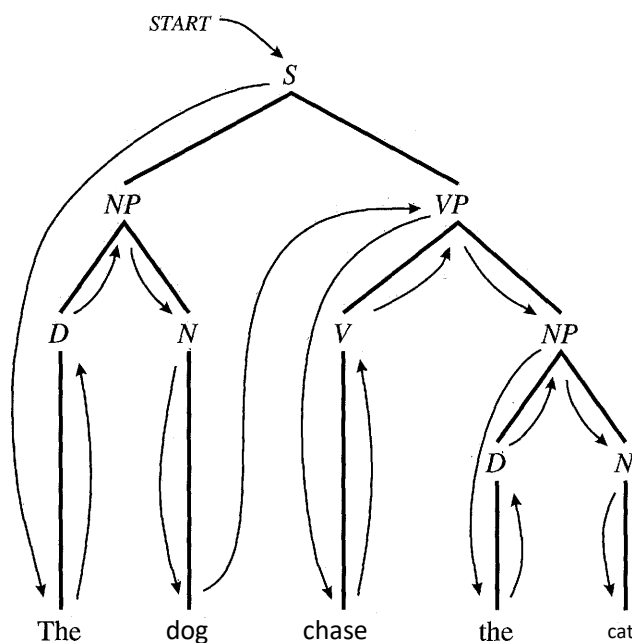


Figure 6.5 Left-corner parsing. Note that some nodes are visited twice, once working top-down and once working bottom-up.

Like a top-down parser, a left-corner parser is always expecting a particular constituent and therefore knows that only a few of the grammar rules are relevant. This can give it an efficiency advantage over straight bottom-up parsing.

But like a bottom-up parser, the left-corner parser can handle rules like $A \rightarrow AB$ without looping, because it starts each constituent by accepting a word from the input string.

6.4.2 The Algorithm

The algorithm consists of two parts: accepting and identifying a word, and then completing the constituent.

To parse a constituent of type C:

1. Accept a word from the input string and determine its category. Call its category

2. Complete C. If $W = C$, you're done. Otherwise,

- Look at the rules and find a constituent whose expansion begins with *W*. Call that constituent *P* (for “phrase”). For example, if *W* is Determiner, use the rule *NP D N* and let *P* be Noun Phrase.
- Recursively left-corner—parse all the remaining elements of the expansion of *P*. (This is the top-down part of the strategy.)
- Last, put *P* in place of *W*, and go back to the beginning of step 2 (i.e., start over trying to complete *C*).

All of this is easier to describe in Prolog than in English. Rules will be represented as they were with our top-down parser:

```
rule(s, [np, vp]).
rule(np, [d, n]).    % etc.

word(n, dog).   word(n, dogs).
word(n, cat).   word(n, cats).    % etc.
```

and so on. To tackle any constituent, the parser first accepts a word, then completes the constituent, thus:

```
% parse(+C,+S1,-S)
%   Parse a constituent of category C
%   starting with input string S1 and
%   ending up with input string S.

parse(C, [WordIS2], S)
    word(W, Word),
    complete(W, C, S2, S).
```

Again, we have `parceptist`, which parses a list of constituents:

```

parse_list1[CICs],S1,S) -
    parse(C,S1,S2),
    parse_list(Cs,S2,S).

```

```

parse_list([],S,S).

```

The only nontrivial procedure is `complete`, defined as follows:

```

% complete(+W,+C,+S1,-S)
%   Verifies that W can be the first subconstituent
%   of C, then left-corner-parses the rest of C.

complete(C,C,S,S).    % if C=W, do nothing.

```

```

parse_list(Rest, S1, S2),
complete(P, C, S2, S) .

```

Exercise 6.4.2.1

Get this parser working with all the grammar rules from Figure 6.1 except $D \rightarrow B$.

Exercise 6.4.2.2

Add arguments to the nodes in the rules so that this parser will build a representation of the tree and enforce subject-verb number agreement. (Easy—just copy the modified rules from Exercise 6.2.0.3.)

Exercise 6.4.2.3

Modify this parser so that it builds a representation of the tree by itself, without requiring any arguments in the rules.

Exercise 6.4.2.4

(*Not for courses that skipped Chapter 5.*) Combine Mini-GULP with this parser. Account for number agreement of subject and verb, and of determiner and noun, using GULP feature structures. (Easy—copy the rules from Exercise 6.2.0.4.)

Exercise 6.4.2.5

Modify `comp1 ed e` so that it can handle null constituents in the portion of the tree that it encounters top-down. Then modify the grammar so that the only rule expanding NP is $NP \rightarrow r D N PP$ where $PP B$. In your modified parser, `rule(pp, (1))` should allow the parser to skip the PP without trying to parse it.

Exercise 6.4.2.6

What does the left-corner parser do when asked to “generate a sentence” by a query such as the following?

```

?- parse(s, What, []).

```

Recall that this query does indeed generate a sentence with the original top-down parser.

6.4.3 Links

As described so far, the left-corner parser is only partly able to handle rules that introduce null constituents, such as $D \rightarrow B$.

If the null constituent is encountered while parsing top—down, there is no problem. The left—corner parser works just like a top—down parser. This is the situation with rules of the form

$$A \rightarrow B C$$

$$C \rightarrow \emptyset$$

because the C gets parsed top-down. All that is necessary is to tell `parse/3` that if it's looking for a C, it can simply skip ahead without parsing anything. This could be accomplished by encoding the rules as

```
rule(a, [b, cJ] ).
rule(c, []) .
```

and adding the following clause to `parse`:

```
parse(C, S2, S)
    rule(W, []),
    complete(W, C, S2, S) .
```

That is: "If W is a category that can be a null constituent, then it is permissible to 'complete' a W at any time by doing nothing."

Unfortunately, this is not what we need for English. Our best-established null constituent is the null determiner, which occurs at the *beginning* of a noun phrase and will therefore be parsed bottom-up. That is, the rules are

$$\begin{array}{rcl} S & \rightarrow & NP VP \\ NP & \rightarrow & D N \\ D & \text{--r} & \text{7} \end{array}$$

and these tell the parser to accept the null determiner as the very first step of parsing an S or NP.

With rules like these, the parser loops. The problem arises whenever it tries a parse that does not succeed. Suppose the parser is looking for a VP but the VP isn't there. This means the parser can't accept the first word of the VP, which could only be a verb. Instead, it "accepts" a null determiner and hypothesizes the structure:

NP

D

N

Next it tries to complete the NP by parsing the N top-down. But suppose the N isn't there

course this will again fail the same way, and the same action will occur over and over. The problem is that, like a bottom-up parser, the left-corner parser is allowed to “accept” null constituents anywhere it wants to. The solution is to constrain the parser by adding a table of rINKs. Besides solving the null-constituent problem, the links make the parser more efficient.

The table of links specifies what kinds of constituents can appear at the beginning of what. For example, from the rules

<i>S</i>		<i>NP VP</i>
<i>NP</i>	$\rightarrow$	<i>D N</i>
<i>VP</i>		<i>V NP</i>

we can infer the links:

```
link(np,s) .
link(d,np) .
link(d,s) .
link(v,vp) .
link(x,x) .
```

The last of these says that any constituent can begin with itself; it applies when, for example, the parser is looking for an N and is also about to accept an N.

We modify `parse / 3` so that there has to be a link between the constituent it is looking for and the word it is trying to accept, thus:

```
parse(C, [Word|S2j,S)
    word(W,Word) ,

    link(W,C) ,
    complete(W,C,S2,S) .

parse(C,S2,S) :-
    rule(W,[]), % for null constituents
    link(W,C) ,

    complete(W,C,S2,S) .
```

This solves the problem because now the parser can only accept a null determiner if

it's already looking for an S or an NP. It can't "accept" null determiners willy-nilly in arbitrary locations.

A left—corner parser can still loop on a set of rules of the form

$$\begin{array}{l} A \quad \quad B A \\ B \rightarrow \epsilon \end{array}$$

but in this case the looping is really the fault of the rules, not of the parser. These rules really do specify an infinite number of different parses for each string: an A can consist

of a null B followed by another A, or of two null Bs followed by another A, or of three null Bs followed by another A, ad infinitum. So infinite backtracking is really the correct thing to do with a grammar of this form. When the grammar specifies an infinite number of parses, one can hardly fault the parser for trying to find them all.

Exercise 6.4.3.1

Add $D \rightarrow r$ 8 to your left—corner parser and observe what happens. Try to parse both grammatical and ungrammatical sentences.

Exercise 6.4.3.2

Construct a table of links for the entire grammar in Figure 6.1. Add the table of links to your parser and demonstrate that the entire grammar can now be parsed.

Exercise 6.4.3.3

Define a predicate `generate_links / 0` that looks at the rules in a grammar, generates the links automatically, and asserts them into memory.

Exercise 6.4.3.4

Even in grammars without null constituents, links can make the parser more efficient. Give an example of how this is so.

6.4.4 BUP

Perhaps the best-known implementation of left—corner parsing in Prolog is BUP, developed by Matsumoto, Tanaka, Hirakawa, Miyoshi, and Yasukawa (1983).

In BUP, each PS rule goes into Prolog as a clause whose head is not the mother node, but rather the leftmost daughter. Thus

$$NP \quad D N PP$$

becomes

$d(C, S1, S) \text{ :- } \text{parse}(n, S1, S2), \text{parse}(pp, S2, S3), \text{np}(C, S3, S) .$

That is: “If you’ve just completed a D, then parse an N and PP. Then call the procedure for dealing with a completed NP.” Here C is the higher constituent that the parser is trying to complete, and S1, S2, S3, and S are the input string at various stages.

In addition to a clause for each of the PS rules, BUP needs a “terminating clause” for every kind of constituent, for example:

$\text{np}(\text{np}, S, S) .$

$n(n, S, S) .$

$d(d, S, S) .$

$\text{vp}(\text{vp}, S, S) .$

you were looking for, you're done."

BUP gets its efficiency from the fact that the hard part of the search—i.e., figuring out what to do with a newly completed leftmost daughter—is handled by Prolog's fastest search mechanism, namely the mechanism for finding a clause given the predicate.

Figure 6.6 shows a complete, working BUP parser, without links. Note that, as required by most Prologs, all the clauses with a particular predicate are grouped together. Matsumoto and colleagues use the names `goal` and `dict` for the predicates that we have called, respectively, `perse` and `word`. Their full BUP implementation includes

```
% BUP left-corner parser, without links, without chart

% parse(+C,+S1,-S)

% Parse a constituent of category C starting with input
% string S1 and ending up with input string S.

parse(C,S1,S)
    word(W,S1,S2),
    P ..    ,C,S2,s],
    call(P).

% PS rules and terminating clauses

np(C,S1,S) :- parse(vp,S1,S2), s(C,S2,S).      % S --> NP VP
np(np,X,X).

d(C,S1,S) :- parse(n,S1,S2), np(C,S2,S).      % NP --> D N
d(d,X,X).

v(C,S1,S) :- parse(np,S1,S2), vp(C,S2,S).      % VP --> V NP

s(s,X,X).
```

```
pp(pp,X,X) .  
n(n,X,X) .  
  
% Lexicon  
  
word(d,[thelX],X) .  
word(n,[doglX],X) .  
word(n,[catlX],X) .  
word(v, chaseslX],X) .    % etc.
```

Figure 6.6 A working BUP parser.

links and a chart (see next section). For subsequent developments, see also Okunishi et al. (1988).

Exercise 6.4.4.1

Add rules to the parser in Figure 6.6 to cover the complete grammar from Figure 6.1, except for *D Z*. Demonstrate that the parser works.

Exercise 6.4.4.2

In your parser from the previous exercise, add arguments to nodes to enforce subject-verb number agreement and build a representation of the tree during parsing.

Exercise 6.4.4.3

Add links to BUP by putting calls to `find/2` in each phrase structure rule as well as in the clause for goal itself.

Exercise 6.4.4.4 (small project)

The slow part of BUP is now the execution of `find/2` and `call/1` in `parse`. Eliminate them. This requires major rearranging because now `W` (in `parse`) cannot be the predicate of the procedure that gets called; instead it must be an argument. Use first-argument indexing to good advantage. Compare the speed of the old and new parsers.

Exercise 6.4.4.5 (project)

Write a program that accepts a set of PS rules, in whatever notation you find most convenient, and generates a BUP parser for them.

6.5.1 The Key Idea

Consider the rule

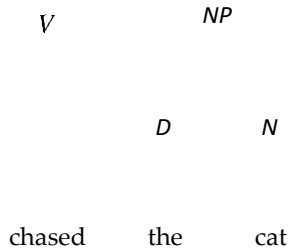
$$VP \rightarrow V NP (PP)$$

This goes into DCG as two rules:

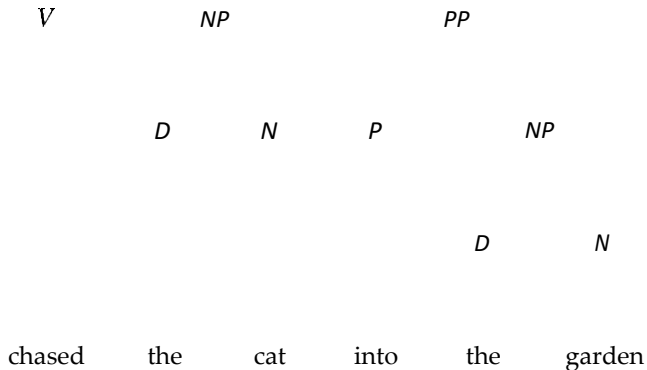
`vp --> v, np.`

`vp --> v, np, pp.`

Now consider the query:



and then realizes it hasn't used up all the words. So it backs up, forgets all the work it has just done, chooses the second VP rule, and *parses the same structure* ngani. Then it goes ahead and parses the PP, giving:



Crucially, the parser had to parse the same V and NP twice. And that could have been a lot of work—recall how complicated an NP can be.

A CHART PARSER or **TABULAR PARSER** is a parser that remembers substructures that it has parsed, so that if it has to backtrack, it can avoid repeating its work. For example, the first time through, a chart parser would make a record that *the ant* is an NP. The sec— and time through, when looking for an NP at the beginning of *the cat into the garden*, it would look at its records (the CHART) before using the NP rule. On finding *SNP the cat*] in the chart it would not have to work through the process of parsing the NP.

6.5.2 A First Implementation

To save each constituent, the parser must record:

- what kind of constituent it is;
- where it begins; and
- where it ends.

One way to represent this information is to store, in the knowledge base, clauses such as:

```
chart(np, [the, cat, into, the, garden], [into, the, garden]) .
```

This means: “If you remove an NP from the beginning of the end *into the garden* you’re left with *into the garden*.” That is, *the cat* is a noun phrase.

The two lists represent positions in the input string. This is not the most concise or efficient way to represent positions, but it is the most readable to human eyes, and we will stick with it through most of this chapter.

Now let’s modify our original top-down parser (Section 6.2) to use a chart. To do this, we modify `parse / 3` so that it looks like this:

```
parse(C, [WordIS], S)
    word(C, Word) .

parse(C, S1, S)
    chart(C, S1, S) .

parse(C, S1, S)  -
    rule(C, Cs) ,
    parse_list(Cs, S1, S) ,
    asserta(chart(C, S1, S)) .
```

The first clause deals with individual words, as in the original top—down parser. It’s faster to look up single words in the lexicon than to try to get them into the chart.

The second clause says, “If what you’re looking for is already in the chart, don’t parse it.” The third clause, using `rule`, is like the original except that it asserts a fact into the chart at the end.

Also, we need a way to discard the chart before starting a new sentence:

```
clear_chart :- abolish(chart/3).
```

Otherwise the chart would become cluttered with obsolete information. So now the usual query to parse a sentence looks like this:

```
?- clear_chart,  
   parse(s,[the,dog,chased,the,cat,near,the,elephant], []).
```

Finally, there is a minor technical problem to be dealt with. Normally, a chart parser will look at the chart before any clauses of `chart` have been asserted. In some Prologs, this causes an error message, which can be prevented by including, at the beginning of the program, a line such as

```
- unknown(_,fail).           % in Quintus Prolog
```

to tell the **Prolog system that queries** to nonexistent **predicates** should simply fail, rather than **raising** error conditions. It is **not sufficient to declare** `charL` as dynamic, because when `clear_char t` **abolishes** `char t`, it will abolish the **dynamic declaration** too.

Exercise 6.5.2.1

Get the chart parser working and use it to parse *Ne dug clizises rlir czir nenr rhe elephant*.

What information is in the chart at the end of the parse?

Exercise Inf.2.2

Modify your chart parser to display a message whenever the second clause of `parse` succeeds.

This will let you know when the chart is actually saving the parser some work. Parse the

dog chases the cat near the elephant again. What output do you get?

Exercise US.2.3 (small project)

Chart parsers need not be top—down. Implement a bottom—up (shift—reduce) chart parser.

6.5.3 Representing Positions Numerically

We noted that storing whole lists in the chart, as in

```
chart(np, [the, cat, into, the, garden], [into, the, garden]) .
```

is inefficient. The lists represent positions in the input string; they can be replaced by word counts (0 for the beginning of the string, 1 for the position after the first word, and so on), so that chart entries look like this:

```
chart(np.0.2) .
```

That is: ‘be NP begins when 0 words have been accepted, and ends when 2 words have been accepted.’ Numbers are more efficient than lists because they are smaller, can be compared more quickly, and can be distinguished by first-argument indexing.

If we do this, we can get rid of the input “string” altogether, and replace it by a set of facts about what words are in what positions. For example:

```
c(the,0,1) .
```

```
c(dog,1,2) .
```

```
c(eees,2,3) .
```

```
c(the,3,4) .
```

```
c(cat,4,5) .
```

```
c(near,5,6) .
```

```
c(the,6,7) .
```

```
c(elephant,7,8) .
```

It's quicker to look these up than to repeatedly pick a list apart.

All this requires almost no change to the parser. The only difference is that `parse/3` uses a slightly different method to accept a word:

```

parse(C, S1, S)

    chart(C, S1, S) .

parse(C, S1, S)

    c(Word, S1, S),    % this is the only change
    word(C, Word) .

parse(C, S1, S) -
    rule(C, Cs) ,
    parse_list(Cs, S1, S) ,
    asserta(chart(C, S1, S)) .

```

In one brief test, making this change sped up the parser about 30% with a simple grammar. If the grammar is more complex, the speed—up could be greater. On the other hand, it takes time to convert each sentence into a set of `c/3` clauses.

Having introduced this technique, we will now put it aside, because in the rest of this chapter, it is much more important for the examples to be readable than for them to run fast. Converting the various chart parsers to use numeric positions is left as an exercise for the implementor.

Exercise 6.5.3.1

Get this parser working and measure the speed-up on your computer. To do this, you will probably have to parse the same sentence hundreds or thousands of times; be sure to clear the chart in between.

6.5.4 Completeness

As implemented so far, the chart parser can remember what it found, but it can't remember what it failed to find. For example, it can remember that *fAr cat* is an NP, but it can't remember that *r/te cat* is not a PP. This means that with a complex grammar, the parser can waste more time than it saves.

To parse any constituent, the parser first looks at the chart. If the chart doesn't contain the constituent that it's looking for, then the parser proceeds to do all the work that it would have done if there had been no chart in the first place. In such a case the chart doesn't save it any work.

This situation arises mainly with fairly large grammars. Here is a simple example. Consider the phrase—structure rules

<i>VP</i>		<i>V NP (PP) (Adv)</i>
<i>NP</i>	$\rightarrow$	<i>D N [PP]</i>
<i>PP</i>		<i>P NP</i>

Suppose the parser is parsing the sentence 'the boy yesterday'. Two different rules allow a PP to come after *boy*. Accordingly, the parser will make two attempts to find a PP at that position, and both attempts will fail. Each attempt will consist of a fair bit of work (invoking *PP P NP*, then looking at lexical entries for *P*, and failing to find *yesterday* among them). If, the first time through, the chart could record that a PP is definitely *not* present at that position, the second attempt to find one would be unnecessary.

There is also another reason why the chart needs to store negative information: unwanted alternatives. Every parse that can be found in the chart can also be found without using the chart. This means that, upon backtracking, the parser can often get the same constituent more than one way, and the number of alternatives to try grows exponentially.

A much better approach is to use the chart only if it is *complete* for a particular constituent type in a particular position. That is, if you're looking for an NP at the beginning of *the cat near the elephant*, use the chart only if it is known to contain all possible NPs in that position. Otherwise, ignore the chart and parse conventionally. And if, when parsing conventionally, you fail to find what you're looking for, then assert that the chart is complete for that constituent in that position, because there are no more alternatives to be found.

We can record completeness by asserting facts such as

```
complete(np, [the,cat,into,the,garden]).
```

Modified to work this way, the parser looks like this:

```
parse(C,[Word1S],S)
    word(C,Word).

parse(C,S1,S)
    complete(C,S1),

    chart(C,S1,S).

parse(C,S1,S) :-
    rule(C,Cs),
    parse_list(Cs,S1,S2),
    asserta(chart(C,S1,S2)),
    S2 = S.
```

```
parse(C,S1,_)
  asserta(complete(C,S1)),
  fail.
```

The first clause is as before. The second clause is as before except that it requires the chart to be complete, and if the chart is complete, it performs a cut so that the

subsequent clauses won't be used. The third clause is as before except that it passes 52 to `parse` `list` uninstantiated, in an attempt to get as many alternatives into the chart as possible, and then checks afterward whether `s2 = S`. The last clause asserts that the chart is complete if all the other clauses have failed.

It is also necessary to modify `clear_chart` to erase `completed` as well as `chart`, thus:

```
clear_chart :- abolish(chart/3), abolish(complete/2).
```

Exercise 6.5.4.1

Demonstrate the problem of unwanted alternatives. Using the parser from Section 6.5.2 (without completeness checking), execute the query:

```
?- parse(s, [the, dog, chases, the, cat, near, the, elephant], []),
   write(y),
   fail.
```

How many `y`'s do you get? Why?

Exercise 6.5.4.2

Modify the parser to include the completeness check and try the same query.

Exercise 6.5.4.3

Why can't there be a cut in the second clause of `parse` if the parser does not check for completeness? That is, in the parser in Section 6.5.2, why couldn't that clause have been as follows?

```
parse(C, S1, S) :- chart(C, S1, S), '.
```

(Hint: Parse *the dog chases the cat near the elephant.*)

Exercise 6.5.4.4

As implemented, the chart parser tries to parse single-word categories (N, V, etc.) with `chart` and `rule` as well as with `word`. We can make `parse` faster by changing its first clause to:

```
parse(C, [Word|S], S)
    word(C, _),

    word(C, Word) .
```

What does this do to the search process? How does it affect the way we have rendered DZ?

If nodes have arguments, chart parsing runs into another problem. Suppose the parser is looking for `np (X)` and the chart contains `np ( singular )` at the right position. Should the parser use this chart entry? Definitely not. Doing so would instantiate `X` to the value `singular`, which might be wrong.

To illustrate this problem is not particularly easy; there are no simple examples in English. It's the kind of thing that crops up suddenly in the middle of a large project after dozens of other rules have worked correctly.

For the determined reader, however, here is a somewhat contrived example. Recall that one way to handle verb subcategorization is to assign the verbs to numbered classes. Consider, then, the following PS rules. For legibility, they are written in DCG format, although in our actual parser they would be clauses of `rel e / 2`.

```
vp --> verbal(0).

vp --> verbal(X), rest_of_vp(X).

rest_of_vp(1) --> np.
rest_of_vp(2) --> np, np.    % etc.

verbal(X) --> v(X).

v(0) --> [sleep].
v(1) --> [see].
v(2) --> [give]. % etc.
```

Assume the usual expansions of `np`. Now execute the query:

```
?- clear chart, parse(vp, [see,the,dog], []).
```

This parse should succeed, but it doesn't. The first VP rule looks for a `verbal(0)` and, of course, doesn't find it. So the chart is marked as complete for `verbal(0)` in that position:

```
complete(verbal(0), [see,the,dog]).
```

Now the second VP rule looks for `verbal(X)` in the same position, with `X` uninstantiated. And `verbal(X)` matches the stored `verbal(0)`, so the parser thinks the chart is complete for `verbal(X)` as well.

The problem, of course, is that `verbal(X)` with the stored `verbal(0)` is the wrong thing to do. The parser should instead check whether the stored category SUBSUMES the one it is looking for. We say that term A subsumes term B if and only if:

- A can be unified with B;
- When this is done, B is no more instantiated than it was before.

For example, $f(X, b)$ subsumes $I(a, b)$, but $f(a, b)$ does not subsume $f(X, b)$ because performing the unification would instantiate X .

The Quintus Prolog built-in predicate `subs espcbk / 2` succeeds if its first argument subsumes its second argument. With subsumption checking added, the second clause of `parse` becomes:

```
parse(C,S1,S)
    complete(C0,S1),
    subsumes_chk(C0,C),
    C0 = C,

    chart(C,S1,S).
```

Figure 6.7 gives an implementation of `subsume spchk` for other Prologs.

```
% Subsumption checker.
% Based on code by R. A. O'Keefe
% in shared Edinburgh (later Quintus) library.

% subsumes_chk(?T1,?T2)
% Succeeds if term T1 subsumes T2, i.e.,
% T1 and T2 can be unified without further
% instantiating T2.

subsumes_chk(T1,T2) -
    \+ ( numvars(T2), \+ (T1 = T2) ).

% numvars(+Term)
% Instantiates each variable in Term to a unique
% term in the series vvv(0), vvv(1), vvv(2)...
```



```
numvars(Term) :- numvars_aux(Term,0,_).
```

```
numvars_aux(Term,N,N) :- atomic(Term),'
```

```
numvars_aux(Term,N,NewN)
```

```
    var(Term),!,
```

```
    Term = vvv(N),
```

```
    NewN is N+1.
```

```
numvars_aux(Term,N,NewN) -
```

```
    Term =.. List,
```

```
    numvars_list(List,N,NewN).
```

```
numvars_list([],N,N).
```

```
numvars_list([Term|Terms],N,NewN)
```

```
    numvars_aux(Term,N,NextN),
```

```
    numvars_list(Terms,NextN,NewN).
```

Demonstrate the problem. Use the parser from the previous section with the grammar

this section, and show what goes wrong upon trying to parse *see the dog*.

Exercise 6.5.5.2

Solve the problem by adding subsumption checking, and demonstrate that your modified parser works correctly.

Exercise 6.5.5.3

Why does `parse` use the subsumption checker when calling `complete` but not when calling `starC`?

Exercise 6.5.5.4

With the grammar in this section, the modification suggested in Exercise 6.5.4.4 no longer works correctly. Explain why, and make the necessary changes.

Exercise 6.5.5.5

Modify the grammar by adding arguments to nodes to account for subject-verb number agreement and to build a representation of the tree. (This is simple; just copy the modified grammar from Exercise 6.2.0.3.)

Exercise 6.5.5.6

(Not for courses that skipped Chapter 5.) Combine Mini-GULP with this parser. Account for number agreement of subject and verb, and of determiner and noun, using GULP feature structures. (Easy—copy the rules from Exercise 6.2.0.4.)

i.6 EARLEY'S ALGORITHM

6.6.1 The Key Idea

Earley (1970) introduced a chart parsing algorithm with the following characteristics:

- It parses n -word sentences in, at most, time proportional to n^3 , which is near the best possible performance.
- It handles null constituents correctly.
- It does not loop on left-recursive rules $A \rightarrow A B$.

- It uses a combination of top-down parsing (“prediction”) and bottom—up parsing (“completion”).
- It is an ACTIVE CHART PARSER, i.e., the chart stores work in progress as well as completed work.
- The parser does not backtrack; instead, it pursues all alternatives concurrently, and at the end, all alternative parses are in the chart.

The key idea of Earley's algorithm is that the chart can store unfinished constituents as well as constituents that have been completely parsed. To understand how this is possible, recall that in DCG, the rule $S \rightarrow NP VP$ goes into Prolog as:

```
s(S1,S) :- np(S1,S2), vp(S2,S).
```

Now consider the query

```
?- s([the,dog,barked],[J]).
```

Imagine combining the query with the rule, like this:

```
s([the,dog,barked],fJ) :- np([the,dog,barked],S1), vp(S1,[]).
```

This could be an Earley chart entry. Now imagine that the parser has processed the NP *the dog*. Then this same rule can be simplified to:

```
s([the,dog,barked],[J]) :- vp([barked],[J]).
```

That is: If *barked* is a VP, then *the dog barked* is an S. And after the VP is parsed, the rule simplifies further to:

```
s([the,dog,barked],[J]).
```

which indicates that the entire sentence has been parsed.

6.6.2 An Implementation

In Earley's original notation, these three chart entries just mentioned would have

the form:

<i>S</i>	<i>•NP VP</i>	<i>0</i>	<i>0</i>
<i>S</i>	<i>NP• VP</i>	<i>0</i>	<i>2</i>
<i>S</i>	<i>NP VP•</i>	<i>0</i>	<i>3</i>

The dot in the PS rule indicates which subconstituent is to be parsed next. The two numbers are word counts; they indicate, respectively, the position of the beginning of the *S*, and the position of the dot, relative to the input string.

To implement Earley's algorithm in Prolog, we will store chart entries in yet a different form, like this:

```
chart(s, [the,dog,barked], [np,vp], [the,dog,barked]).  
chart(s, [the,dog,barked], [vp], [barked]).
```

- The constituent originally being sought;
- The position at which that constituent was supposed to begin;
- The subconstituents (GOArs) currently being sought (we'll call the first goal in each list the coRRENT GOAL); and
- The cuRRENT POSITioN in the input string, where the current goal should begin.

Obviously, an Earley parser has many current goals, one corresponding to each chart entry at the current position. (The inefficient representation of positions is deliberate; see section 6.5.3.)

Earley's algorithm parses entirely by manipulating the chart. This means that the parse procedure is very simple—put the input string and initial goal into the chart, let the parser do its thing, and then see whether it has produced a successful parse:

```
parse(C,S1,S) :-
    clear_chart,

    store(chart(start,S1,[C],S1)),
    process(SI),

    chart(C,S1,[],S).
```

Here process steps through the input string, calling the three parts of the parser at each position:

```
process([]) :- '

process(Position)
    predictor(Position),
    scanner(Position,NewPosition),
    completer(NewPosition),
    process(NewPosition).
```

The three parts of the parser itself are:

- The REDICTOR, which looks for rules that expand current goals, and uses them to create new goals;
- The scANNER, which accepts a word from the input string and uses it to satisfy current goals; and

- The couPLETER, which looks at the output of the scanner and determines which, if any, larger constituents have been completed.

All three of these produce chart entries. For example, given

```
chart(s, [the,dog,barked], [np, vp], [the,dog,barked]).
```

the predictor will use $NP \rightarrow D N$ to produce `chart(np, [the,dog,barked], [d,n], [the,dog,barked])`. Then the scanner will accept *the*, giving: `chart(np, [the,dog,barked], [n], [dog,barked])`.

Now the completer executes, but can't do anything because nothing has been completed; then the predictor executes, but can't do anything because there are no rules expanding

n. So the scanner gets to execute again; it accepts *dog*, giving:

`chart(np, [the,dog,barked], [], [barked])`.

Now the NP is complete and the completer takes note of this fact, modifying the chart entry for S appropriately:

`chart(s, [the,dog,barked], [vp], [barked])`.

In a similar way the predictor, scanner, and completer process the VP. Then the S is complete and the last thing put into the chart is:

`chart(s, [the,dog,barked], [], [])`.

Since S was the original goal and the input string is empty, the parse is complete.

The predictor, scanner, and completer communicate mainly through the chart, except that each of them has to know the current position in the input string (and the scanner changes it). Here is a more precise description of what they do:

Predictor

For each current goal at the current position,
look up all the rules expanding it
and use them to make more chart entries.
(This creates additional current goals at the current
position; do the same thing to all of them.)

Scanner

Accept a word from the input string and determine its category.
Look for all chart entries for the current position
whose active goal is that category.
Make, from each of them, a new chart entry,

Look for constituents that have just been completed, and use them to complete higher-level constituents. (Upon completing a constituent this way, do the same thing with the result.)

Figure 6.8 shows, in detail, the process of parsing *The dog chases the col*. For simplicity, lots of irrelevant syntax rules are ignored in the example. With a larger grammar, there would be somewhat more entries in the chart and more work at each stage.

Exercise 6.6.2.1

By hand, work through the process of parsing *The dogs chase the cat near the elephant* using Earley's algorithm and the grammar in Figure 6.1. Include all chart entries, including those from rules that do not contribute to a successful parse. Show your results in the format of Figure 6.8.

6.6.3 Predictor

The predictor looks at all the chart entries that apply to the current position in the input string, and constructs, from each of them, new chart entries expanding their current goals. This work is done by two predicates, `pr ed i odor` and `pr ed i c L`, both of which use `fail` to make execution backtrack through all possibilities:

```

predictor(Position) -

    chart( , , [GoalI_], Position),    % For every chart entry of this
    predict(Goal, Position),           % kind do all possible

    fail.                               % predictions

predictor( ).                          % then succeed with no further
                                       % action.

predict(Goal, Position)

    rule(Goal, [HIT]),                 % For every rule expanding Goal

    store(chart(Goal, Position,

                [HIT], Position)),    % make a new chart entry and
    predict(H, Position),              % make predictions from it too
    fail.

```

```

predict(_,_).                                % then succeed with no further
                                              % action.

```

There are two predicates here because `pr edic L` has to call itself in order to make further predictions from the chart entries it creates.

The alert reader will notice that `predicL or` looks for chart entries at `Pos i t i` `on`, and `pr edic t`, which it calls, is constantly adding chart entries at the same `Po s i` `C i on`.

Start with:	Scan:
Predict:	Complete: Predict: Scan:
Scan:	Complete:
Complete:	
Predict:	
Scan:	
Complete:	
Predict:	
Scan:	
Complete:	
Predict:	

chart (start , fthe , dog ,
chases , the , cat) , fs) ,
ft he , dog , chases , the ,
cat)) .

Use *S* → *r NP VP* and *NP* → *r D N*.

chart (s , fthe , dog , cha
ses , the , cat) , [
np , vpj , [tbe , dog , chas
es , the , catj) .

chart (np , the , dog ,
chases , the , cat) , (d,n3
 , ft he , dog , chases , the
 , cats) .

Accept *the*.

chart (up , (the , dog ,
chases , the , cat3 , fn3 ,
dog , chases , the , cat3)
 .

N

O

t

h

i

n

g

t

o

d

o

;

no phrase has been completed. Nothing to do;

there are no rules expanding N. Accept *dog*.

chart (np , [the , dog , chases , the , cat] , [] , [chases , the , catj) .

An NP has now been parsed.

chart (s , [the , dog , chases , the , catj , [vp] , [chases , tAe , cat]) .

Use *VP* → *r V* and *VP* → *r V NP*.

chart (vp , [chases , the , cat] , [v] , [chases , thp , cat]) .

chart (vp , [chases , the , catj , [v , npj , [chases , the , catj]) .

Accept *chases*.

chart (vp , chases , the , cat) , f3 , bthe , cat)) .

chart (vp , fchases , the , cat) , (np3 , ft he , cat3) .

According to *VP* → *V*, a VP and hence the S have now been parsed.

chart (s , [the , dog , chases , the , cat] , [4 , [the , catj]) .

chart (start , [the , dog , chases , the , cat] , [j , [the , catj]) .

The other VP rule is still looking for an NP. Expand it..

chart (np , [the , catj , [d , nj , [the , catj]) .

Accept *the*.

chart (np , [the , cat] , [nj , [cat]) .

Nothing to do—no phrase has been completed.

Nothing to do—there are no rules expanding N.

Accept *car*.

chart (np , [the , cat] , L1 , L1) .

Now the NP, and hence the VP and S, have been parsed.

chart (vp , [chases , the , catj , [j , []]) .

chart (s , (the , dog , chases , the , cat) , I .)) .

c
h
a
r
t
(
s
t
a
r
t
,
(
t
h
e
,
d
o
g
,
c
h
a
s
e
s
,
t
h
e
,
c
a
t
)
,
(
)
,
(
)
)
.
A
l
l
d
o

ne.

Figure 6.8 Earley's algorithm in action.

backtracking anyway?

For two reasons. First, `s o re` (which we haven't defined yet) uses `as s er C a`, `as s en z`, and thus the new entries are added before the one presently being looked at. Second, even if `s Core` didn't do this, most Prologs would never see the new entries, because in most Prologs, newly added clauses cannot become alternatives for a query that is already in progress. Upon starting any query, Prolog determines which facts and rules can satisfy it, and any further rules that get asserted during the processing of the query will be ignored.

Exercise 6.6.3.1

Get the predictor working and show that it can execute the first part of the parse in Figure 6.8, up to the point where the scanner is needed. Temporarily use `as s en s` in place of `store`.

6.6.4 Scanner

The scanner is very simple:

```
scanner([WlWordsJ,Words)

    chart(C,PC,[GlGoals],[WIWords]), % for each current goal at
                                     % current position

    word(G,W),                       % if category of W matches it
    store(chart(C,PC,Goals,Words)), % make a new chart entry
    fail.

scanner([_IWords],Words).           % then succeed with no further
                                     % action.
```

Traditionally, the predictor predicts individual words. We do not do this here, because it would add a vast number of unnecessary predictions to the chart. Instead, the scanner determines the category of each word at the time the word is accepted.

Exercise 6.6.4.1

Add the scanner to the predictor that you have already gotten working. Show that your scanner and predictor can execute the first two steps of the parse in Figure 6.8, up to the point where the completer is needed.

Exercise 6.6.4.2

Why doesn't second clause of `s canner` have anonymous variables in place of `Words`?

6.6.5 Completer

The completer looks at all chart entries at the current position that have no goals—i.e., all completed constituents—and tries to use them to complete larger constituents. Normally

the scanner produces one completed constituent (a single word, such as tN *dog*), and the completer tries to use this to complete something larger, such as an NP.

Again, there are two predicates, because the completer has to call itself on its own output:

```
completer(Position)
  chart(C, PC, [], Position), % For every chart entry with no
  complete(C, PC, Position), % goals, complete all possible
  fail. % higher constituents,

completer(_). % then succeed with no further
              % action.

complete(C, PC, Position)
  chart(C0, PC0, [CiGoals], PC), % For every constituent that can be
  store(chart(C0, PC0, % completed make a new chart entry,
                    Goals, Position)), %
  Goals == [], % then fail here if Goals not empty,
               % or process new entry the same way
  complete(C0, PC0, Position), %
  fail.

               % then succeed with no further
complete( , _, _). % action.

               %
```

The difference is that, unlike `predicC`, `complete` does not *always* call itself on its own output—it does so only if its output is itself a chart entry with no goals, and hence a completed constituent. Recall that the completer is responsible for creating

```
chart(s, [the,dog,chases,the,catJ,[vp],[chases,the,cat]]).
```

```
when the NP is finished and then chart(s,  
[the,dog,chases,the,cat],[],[ ]).
```

when the VP is finished. Only the second of these completes a constituent and thereby justifies a recursive call to the completer.

Exercise 6.6.5.1

Get the completer working and add it to the predictor and scanner that you have already implemented. Test it. Use a grammar that does not include any left-recursive rules.

6.6.6 How Earley's Algorithm Avoids Loops

We haven't defined `st` ore yet. Its definition is the key to the way Earley's algorithm

```
store(chart(A,B,C,D))
\+ chart(A,B,C,D),
asserta(chart(A,B,C,D)).
```

Consider how the predictor handles the rules

$$NP \rightarrow DN$$

$$NP \rightarrow NP \text{ Conj } NP$$

when parsing *the dog and the cat*. At the start, there is no way to tell which rule applies, so the predictor makes two chart entries:

```
chart(np, [the,dog,and,the,cat], [d,n], [the,dog,and,the,cat]).
chart(np, [the,dog,and,the,cat], [np,conj,np], [the,dog,and,the,cat]).
```

Recall that the predictor is supposed to apply to its own output, i.e., the chart entries that it generates. The first of these chart entries has *d* as its current goal; *d* is not a phrase; so no further predictions are made from it. The second chart entry has current goal *up*, which can be expanded by either of two rules, giving two more chart entries:

```
chart(np, [the,dog,and,the,cat], (d,n), [the,dog,and,the,cat]).
chart(np, [the,dog,and,the,cat], [np,conj,np], [the,dog,and,the,cat]).
```

But these entries are *already* in the chart. This means that *sCore* fails, and the predictor doesn't get to call itself recursively on them. Thus the rule is blocked.

To put it more succinctly: *Earley's algorithm will never predict a constituent at a position, if it has already predicted the same constituent at the same position.*

But the completer is welcome to use the same chart entry more than once when completing a recursively embedded phrase. Thus long phrases such as *the dog and the cat and the elephant* are parsed with no problem.

Exercise 6.6.6.1

Determine (by hand or using the computer) the chart entries for parsing *the dog and the cat and the elephant*. Show them in the order in which they are added to the chart.

6.6.7 Handling Null Constituents

As shown, our parser still has trouble with the rule $D \rightarrow B$, which we represent as

```
rule(d, []).
```

because the predictor insists that the second argument of `rule` be a nonempty list.

We have two choices. We can recast $D \rightarrow B$ as `word(d, [])` and modify the scanner to handle it; or we can leave it as it is and modify the predictor.

We choose the first of these, and add a clause for `predict`:

```
predict(Goal, Position) -
    rule(Goal, []),

    store(chart(Goal, Position, [], Position)),
    complete(Goal, Position, Position),

    fail.
```

This comes between the two clauses already defined. This clause generates a new chart entry in the same way as the first clause, but then it calls the completer—necessary because a constituent has been completed, of course, and the completer now needs to work on it at *this* position in the input string. Normally the completer would not get to work until after the scanner had advanced the current position to the next word.

Exercise 6.6.7.1

Get an Earley parser working which includes the whole grammar in Figure 6.1.

Exercise 6.6.7.2

Add features to the nodes in the grammar to enforce subject-verb number agreement and build a representation of the tree structure (as in Exercise 6.2.0.3).

6.6.8 Subsumption Revisited

Like all chart parsers, Earley's algorithm needs a subsumption check. This turns out to be just a simple modification to `store`: instead of checking for pre-existing chart entries that *match* the new one, check for entries that *subsume* it.

```
store(chart(A, B, C, D))
```

```
\+ (chart(A1,B,C1,D), subsumes_chk(A1,A), subsumes_chk(C1,C)),  
asserta(chart(A,B,C,D)).
```

Here we save some time by looking only at A and C, which are the only parts of the chart entry in which uninstantiated arguments can appear.

Recall that subsume spebk is not built in. If you need to define subsume opchR for yourself but forget to do so, the parser will still work but the chart will contain superfluous entries, because all the calls to subsume spchk will fail.

Exercise 6.6.8.1

Demonstrate the need for a subsumption check in your Earley parser, using the same grammar as in Exercise 6.5.5.1. Then modify your parser to correct it.

Exercise 6.6.8.2

As you have done with all the other parsers, add arguments to the nodes to account for subject-verb number agreement and to build a representation of the tree. (See Exercise

put the book on the table

Figure 6.9 Parse tree with subcategorization lists.

Here we're using DCG notation for clarity, but this last rule is, of course, not legal in DCG, because one of the constituents (X) is a variable. Nevertheless, Earley's

algorithm should be able to handle it, because as soon as the scanner accepts the verb, the value of X will be known.

Earley's algorithm loops for a different reason—the occurrence of the argument Y within the argument $[X!Y]$. Starting with

```
chart(v([], [sees, the, dog], [v([X]), XJ, [sees, the, dog]]).
```

the predictor will generate, in succession,

```
chart(v([X]), [sees, the, dog], [v([X1, X]), X1], [sees, the, dog]).
```

```
chart(v([X1, X1], [sees, the, dog], [v([X2, X1, X]), X2], [sees, the, dog]).
```

```
chart(v([X2, X1, X]), [sees, the, dog], [v([X3, X2, X1, X]), X3], [sees, the, dog]).
```

```
chart(v([X3, X2, X1, X]), [sees, the, dog], [v([X4, X3, X2, X1, X]), X4], [sees, the, dog]
```

ad infinitum, where $X, X1, X2, \dots$ represent the variables that correspond to X on successive invocations of the $v(Y)$ rule.

Shieber (1986) proposes a solution that he calls RESTRiCriON: make the predictor ignore arguments whose values are not fully instantiated. By “ignore” we mean, in the context of our parser, that anonymous variables will be substituted for these arguments. (This makes the predictor overpredict, but no harm results because the spurious predictions are not used further.) With restriction, *sees the dog* is parsed as follows:

```
Predict:  chart(v([], [sees, the, dog], v(_), [sees, the, dog])).
```

```
Scan:
```

```
Complete: chart(v([np], [sees, the, dog], [], [the, dog])).
```

```
          chart(v([np, np], [sees, the, dog], [np], [the, dog])).
```

and so on. Restriction applies only to the predictor; the scanner and completer look back at the rules and fill in the missing arguments. Gerdemann (1989, 1991) explores other uses of restriction.

In this context it is helpful to think of parsing with arguments as a twofold

process—accepting the input string and building the arguments of the nodes. These processes go on concurrently, working, as it were, at right angles to one another. Earley’s algorithm does a good job of loop-proofing the process of accepting the input string, but loops with argument-building are still possible. An important research topic at the moment is the discovery of linguistically motivated constraints on arguments.

Exercise 6.6.9.1

Demonstrate that Earley’s algorithm loops on the grammar shown.

Exercise 6.6.9.2 (small project)

Implement restriction and use it to make Earley’s algorithm handle this grammar success—

The implementation of Earley's algorithm that we have developed is far from optimal. There are numerous ways to improve it from the standpoint of Prolog coding:'

- Represent positions in the sentence as numbers (word counts) rather than lists (as in Section 6.5.3).
- Use first-argument indexing. The parser spends much of its time searching through the chart or through the rules. Organize these so that they have distinctive first arguments that will speed the retrieval of the desired information.
- Reduce the use of `assert`; see what happens if (part of) the chart is kept in a list rather than stored in the knowledge base. This may be beneficial during some kinds of processing but not others.

Another tack is to improve the predictor. Except for the loop-check, the usual Earley predictor works just like a recursive—descent top-down parser. Like all top—down parsers, it makes plenty more predictions than are really necessary. Earley (1970) suggested letting the predictor look ahead to the next entry in the input string, and thereby narrow down the range of choices. Kilbury (1984) replaced the top—down predictor with one that works like a left-corner parser; Leiss (1990) discusses Kilbury's proposal and then presents his own, which works like a left—corner parser with links, thereby avoiding many spurious predictions.

Exercise 6.6.10.1 (project)

Implement a *good* Earley parser in Prolog, making as many improvements as possible.

6.6.11 Earley's Algorithm as an Inference Engine

Recall that when we introduced Earley's algorithm, we presented chart entries as stored Prolog rules and facts. This leads us to suspect that Earley's algorithm could serve as a general way to execute Prolog, instead of the normal search-and—backtracking process. And this suspicion is right. Pereira and Warren (1983) and Pereira and Shieber (1987:199—210) discuss "Earley deduction," which they attribute to an unpublished 1975 note by David H. D. Warren.

The basic idea is that any inference engine can store cEMMAs, or previously proved results, in order to keep itself from repeating work upon backtracking. This is the equivalent of chart parsing pure and simple. An Earley inference engine is loop—proof because it never stores a lemma, nor tries to derive further lemmas from it, if the same lemma, or one that subsumes it, is already in the knowledge base. Thus, with Earley deduction,

```
ancestor(X,Z) :- ancestor(X,Y), ancestor(Y,Z).
```

does not cause the infamous loop.

'In this section I am indebted to Joseph Knapka, who experimented with a number of these proposed improvements.

There are differences between parsing and deduction, of course. In Earley deduction, there is no input string and hence no scanner. Instead, both the completer and the predictor need to look at every clause that is added to the chart.

The predictor looks at each rule newly added to the chart, and expands that rule's first subgoal using every possible rule in the knowledge base, then does the same thing, if possible, to the results. Thus from

```
f(a) :- g(a). (newlyadded)
g(X)    h(X), j(X). (intheknowledgebase)
```

the predictor produces $g(a) :- h(a), j(a)$. and adds it to the chart.

The completer deletes subgoals by resolving them with facts. When a fact is added to the chart, the completer will resolve it against the first subgoal of every possible rule already in the chart, so that for example

```
h(a). (newly added)
k(X) :- h(X), m(X). (already in the chart)
```

yields the new chart entry $k(a) :- m(a)$.

The completer also works on newly added rules, resolving them against facts in the chart or in the knowledge base. If for example

```
k(x) :- h(x), m(x).
```

has just been added and

```
h(a).
```

is already in the chart or the knowledge base, the completer will produce $k(a) :- m(a)$. just as in the previous example.

Figure 6.10 shows this process in action. Clauses are arranged in sets. First the predictor works on a set, producing the next set of new clauses; then the completer

works on all the clauses in that set, producing the next set; then the predictor operates again; and so on.

Earley deduction is slower and requires more memory than ordinary Prolog deduction. Even with simple computations, the chart can become quite large, and the process of searching it, quite time-consuming. Further, even Earley deduction can loop on rules of the form

$$f(X) \text{ :- } f(a(X)) .$$

because neither X nor $f(X)$ subsumes the other, leading to the same problem as with grammars in the previous section.

All this reminds us of the fact that Prolog was never meant to be a perfect inference

	3.	$a(X, Z) :- a(X, Y), a(Y, Z).$	
Query:	4.	$start :- a(a, c).$	
Predictor:	5.	$a(a, c) :- a(a, Y0), a(Y0, c).$	from 4 and 3
	6.	$a(a, Y0) :- a(a, Y11), a(Y1, Y0).$ (Loop checker intervenes here, to prevent deriving a redundant clause from 6 and 3.)	from 5 and 3
Completer:	7.	$a(a, c) :- a(b, c).$	from 5 and 1
	8.	$a(a, c).$	from 7 and 2
	9.	$start.$ (This is a solution to the original query; computation could stop here, or continue in order to look for other solutions.)	from 8 and 4
Predictor:	10.	$a(b, c) :- a(b, Y2), a(Y2, c).$	from 7 and 3
	li.	$a(b, Y2) :- a(b, Y3), a(Y3, Y2).$ (Loop checker intervenes here, to prevent deriving a redundant clause from 11 and 3.)	from 10 and 3
Completer:	12.	$a(b, c) :- a(c, c).$	from 10 and 2
	13.	$a(b, Y2) :- a(c, Y2).$	from 11 and 3
Predictor:	14.	$a(c, c) :- a(c, Y4), a(Y4, c).$	from 12 and 3
	15.	$a(c, Y2) :- a(c, Y5), a(Y5, Y2).$ (Loop checker intervenes.)	from 13 and 3
Completer:		No further action (nothing matches $a(c, Y4)$ or $a(c, Y5)$).	
Predictor:		No further action (no new clauses).	

Figure 6.10 Earley deduction in action. This computation would loop in ordinary Prolog.

and speed. By not storing lemmas or checking for loops, Prolog runs faster and in less memory than more sophisticated inference engines.

Exercise 6.6.11.1 (project)

Implement Earley deduction as an inference engine for Prolog clauses that do not contain cuts. Evaluate the performance of your implementation.

6.7 WHICH PARSING ALGORITHM IS REALLY BEST?

6.7.1 Disappointing News about Performance

Table 6.1 presents a piece of sad news. It shows the times taken by the parsers in this chapter to parse, 100 times, the 24 sentences generated by the grammar

<i>S</i>	$\rightarrow$	<i>NP VP</i>
<i>N</i>		<i>D N</i>
<i>P</i>	$\rightarrow$	<i>V NP</i>
<i>V</i>		<i>P NP</i>
<i>P</i>		
<i>PP</i>		
<i>D</i>		the
<i>P</i>		near
<i>N</i>		dog, cat
<i>V</i>	$\rightarrow$	chases,
	--r	sees

which is a subset of the grammar in Figure 6.1.'

TABLE 6.1 COMPARISON OF SPEED OF VARIOUS PARSERS.

Parser	Time (in seconds) to parse 24 sentences 100 times ⁴	
	ALS Prolog 20-MHz 80386	Quintus Prolog Sparcstation 1+
DCG rules	3.4	0.3
Top—down interpreter	6.0	1.2
Bottom—up (shift—reduce)	38.3	8.4
Left-corner (no links)	12.7	2.6
Left—corner (with links)	12.0	2.5
BUP (no links, no chart)	38.5	8.4

Chart (no completeness check)	47.2	20.5
Chart (with completeness check)	59.3	27.3
Chart (with subsumption check)	71.3	32.2
Earley (no subsumption check)	320.3	144.5
Earley (with subsumpuon check)	989.8	172.0

* CAUTION: These data were obtained with a very small grammar. See text before drawing any conclusions!

The sad news is that as the algorithms get “better” the parsing gets slower and slower. Top—down parsing is very fast; left-corner parsing is second best; but shift-reduce parsing and all the chart parsers are regrettably slow, to the point that Earley’s algorithm with subsumption is intolerable.

The procedure was to parse the set of 24 sentences once (to allocate memory), start timing, parse the

- LACK OF OPTIMIZATION. Recall that the parsers in this chapter were not designed for speed. Only DCG and BUP compile the grammar rules into executable Prolog clauses; the others store the rules as Prolog facts and manipulate them at run time. Further, no attempt has been made to fully exploit first—argument indexing or to use the most efficient data structures.
- SMALL GRAMMARS. These tests were run with a very small grammar. That's one reason chart parsing comes out so badly. In the test sentences, there is little ambiguity and hence little benefit from using the chart. But the cost of constructing the chart is large, especially when it is built using a *sen*. Pereira and Shieber (1987:196—210), Ross (1989:217—246), and Simpkins and Hancox (1990) present chart parsers in which the chart is passed along in arguments of procedures. In any case, the chart would be much more helpful with a larger grammar.

But Shann (1991) reports that even with large grammars, Earley's algorithm remains quite slow; left-corner parsing and Tomita's bottom—up algorithm run much faster.

- SUBSUMPTION. A third factor is that subsumption checking is rather costly, for two reasons. First, the subsumption checker itself is inefficient (except in Quintus, where it is built in). Second, the subsumption check limits the use of first—argument indexing. Instead of looking directly at the chart for an entry that matches the current goal, the subsumption—checking parser must instead retrieve all the chart entries for the current position, one by one, and run them through the subsumption checker. This means that indexing can't prevent some unnecessary entries from being retrieved.

The lesson to be learned is that parsing isn't easy. Both backtracking, and charts that reduce backtracking, are costly. But there is another way to approach this problem. Any nondeterministic parser can be improved by adding an ORACLE OF TABLE that keeps it from trying grammar rules that won't succeed. This is particularly the case with the shift-reduce parser, which otherwise spends too much time searching through all the rules. A parser with an oracle goes through a series of numbered states. In each state, the oracle looks ahead at the next word in the input string and tells the parser what grammar rule to use and what state to go into next. If this process were deterministic, it would be like the LR(1) parse tables commonly used by programming-language compilers (Aho and Ullman 1972:368—399), but as Nilsson (1986) points out, the language need not be deterministically passable; Prolog can still backtrack if it needs to.

Not only does an oracle reduce backtracking, it also enables a shift—reduce parser to handle null constituents $D \rightarrow B$ without looping, because the parser can only “accept”

a null constituent in places where the table allows one. For a clear exposition of shift— reduce parsing with an oracle, see Pereira (1985).

In any case, the data in Table 6.1 should serve as a warning. Not every “efficient” parser is actually fast; testing is always appropriate.

Exercise 6.7.1.1

What is the difference between table-driven (oracle-driven) parsing and tabular parsing?

Exercise 6.7.1.2 (project)

Take one of the parsing algorithms presented in this chapter and make it work as efficiently as possible with a reasonably large grammar. Support your claims with actual tests.

Exercise 6.7.1.3 (project)

Test some published parsers written in Prolog for which efficiency is claimed, and see if they are as fast as DCG with a moderately large grammar.

6.7.2 Complexity of Parsing

Earley's algorithm made history because it proved that phrase—structure parsing could be done in POLYNOMIAL TIME, i.e., in time proportional to n^k , where n is the length of the input string and k is some constant. Further, Earley proved that $k < 3$.

This is important because recursive-descent and shift—reduce parsers take, in the worst case, EXPONENTIAL TIME, i.e., time proportional to k^n (which, for large n , is much greater than n^k),

Recursive—descent and shift-reduce parsers take exponential time because they try out each possible parse tree separately. With some grammars, every string of n words can have k^n different parse trees, of which only the last one can be used in further parsing. Earley's algorithm gets around this limitation by trying all the parse trees concurrently, so that many of its actions apply to more than one parse tree.

Generally, polynomial—time algorithms reflect principled solutions to a problem, while exponential—time algorithms reflect brute—force combinatorial search. Thus, Earley's

algorithm shows that parsing can be done in a principled way, not just by trying all possible combinations one after another.

These famous results are less important than they appear, for several reasons. First, natural—language parsers never face large values of n . Even exponential—time parsers can be quick when n is small, and in real life, sentences of more than 30 words are uncommon. When long sentences do occur, they can usually be broken up into shorter sentences that are joined by words like *and*, and parsable separately.

Second, the n^d and k^n results apply only to the worst CASE, i.e., the situation in which the parser “guesses wrong” as much as possible and uses the right rule only after trying all the wrong ones. It is common for a k^n -time parser to take much less than k^n time on any particular parse, because it is almost certain to guess right some of the time and thereby avoid unnecessary work.

Third, and most importantly, the u^1 result holds only for grammars in which nodes do not have arguments. Barton, Berwick, and Ristad (1987) proved that parsing, in the general case, is NP—COMPLETE—that is, belongs to a class of problems that are believed to take exponential time—if grammars are allowed to have agreement features (i.e.,

On consideration, this is not unreasonable. There *are* sentences which even beings cannot parse. Consider Barton, Berwick, and Ristad's example:

BUFFALO BUFFALO BUFFALO BUFFALO BUFFALO

Give up? The structure is exactly like *Boston cattle bewilder Boston cattle*. that I've told you this, you probably have no trouble parsing it. And this is a NP—complete problems: even though a solution cannot be *found* efficiently, if you find a solution somehow, it can be *verified* quickly. What buffaloes us here is, course, the multiple ambiguity of each of the words.

Progress in developing more efficient parsers, then, apparently depends on things: the careful study of complexity in typical or average cases (not in the worst case, which is admittedly intractable), and the discovery of constraints on natural-language grammars.

Exercise 6.7.2.1

Show that, with the grammar

NP	$\rightarrow$	$D N_0$ (and NP)
NP	$\rightarrow$	$D pl$ (and NP)
D	$\rightarrow$	the
$N,$	$\rightarrow$	sheep, deer, quail (etc.)
pl	$\rightarrow$	Sheep, deer, quail (etc.)

every NP containing n ambiguous nouns (such as *the deer and the quail and the sheep*) has 2^n different parse trees. (Hint: What happens to the number of parse trees every time an ambiguous noun is added to the phrase?)

Exercise 6.7.2.2

Suppose that one parser takes 1.5^n seconds to parse an n -word sentence, while another parser takes $2.5 \times n!$ seconds. For what values of n is the exponential-time parser faster? (Compute some actual numbers and make a table.)

Exercise 6.7.2.3

(For students who know calculus.) Show that, for any constant $k > 1$ and for sufficiently large n , $k^n > n^k$ (Find the value at which $k^n = n^k$ and then compare the rates at which k^n and n^k grow as n increases.)

Exercise 6.7.2.4

Examine some English sentences in an actual text (perhaps this book). How long is the longest sentence that does not break up into conjoined shorter sentences? What can you observe about the statistical distribution of sentence length?

Exercise 6.7.2.5

Consider the sentences:

There are pigs in the pen.

There is ink in the pen.

Clearly, *pen* needs two lexical entries, one with each of two meanings. What could be done to guide the parser to the right lexical entry in each case, so that it doesn't have to try both of them? (Hint: Consider what it is like when you, as a human being, hear and understand one of these sentences.)

6.7.3 Further Reading

Parsing is a big field, and this chapter has sampled only a small part of it. A vast number of different parsing techniques have been developed, some for natural language, some for programming languages, and some for both. Some good general surveys of natural— language parsing are Winograd (1983), Kay (1980), King (1983), Dowty, Karttunen, and Zwicky (1985), and Tomita (1991). On parsing in Prolog, see, besides the references given throughout this chapter, Dahl and Saint-Dizier (1985, 1988), Abramson and Dahl (1989), and Gal, Lapalme, Saint-Dizier, and Somers (1991).

In this chapter we have looked at parsers designed to handle arbitrary sets of PS rules. But human languages have a structure all their own. PS rules in human languages are constrained in ways that are not yet well understood, and human languages involve many phenomena that PS rules by themselves can't handle. Marcus (1978, 1980) inaugurates an important line of research on parsers designed specifically for human languages. Marcus' parser has no backtracking and limited lookahead; it gives special treatment to NP nodes; and its performance on structurally ambiguous sentences is surprisingly humanlike (see also Kac 1982). More recent "principle-based parsers" deduce the rules of grammar (PS rules, case assignment, etc.) from more abstract principles (Berwick, Abney, and Tenny 1991). Tomita (1986), with more modest goals, presents a bottom-up chart parser optimized to deal with the fact that natural-language utterances are short (usually under 30 words) but have a lot of structural ambiguity.

In many languages, word order is partly or completely variable. For example, the Russian translation of *the dog sees the cat* is *sobaka vidit koshku* with the three words in any order depending on the desired emphasis. PS rules do not work well for such languages, and a variety of other approaches have been taken; see Abramson and Dahl (1989:150—153), Covington (1990), and Kashket (1991).

7.1 THE PROBLEM OF SEMANTICS

Semantics is the level at which language makes contact with the real world. This means that semantics is at once the most important part of natural language processing, and the most difficult. Many problems in computational semantics have, as yet, no widely accepted solutions. In principle, an entirely adequate theory of semantics would require a complete theory of human thought.

The goals of this chapter are more modest. In it, we will focus on how to translate English into Prolog or something close to it, mainly in order to answer database queries. Even though Prolog is not powerful enough to represent human knowledge as a whole, it is adequate and convenient for the kinds of information commonly stored in computers. Practically all computer databases map onto Prolog in a simple way.

The techniques used here are somewhat *ad hoc*, but there is a strong emphasis on the underlying logical theory. The model-theoretic semantics is largely based on the Discourse Representation Theory of Kamp (1981; for a readable exposition see Spencer-Smith 1987, and for an implementation, Covington, Nute, Goodman, and Schmitz 1988).

7.2 FROM ENGLISH TO LOGICAL FORMULAS

7.2.1 Logic and Model Theory

To describe the meanings of natural-language utterances, we need a precise way to describe the information that they contain. We can get this from logic and set theory; it’s called **MoDEL—THEORETIC SEMANTICS**. It relies on **MODELS**, which are precisely defined knowledge bases.

Consider a simple formula such as *chases fido, felix* (‘Fido chases Felix’). This formula is part of a logical language. A **MODEL** for the language consists of a **DoMAIN D**, which is the set of individual people and things that can be talked about, plus an **INTERPRETATION FUNCTION I** which maps everything in the language onto something in the domain. Specifically:

- I maps logical constants (proper names) onto individual members of D. For in— stance, I(fido) is Fido.
- I maps predicates onto sets of tuples of elements of D. For example, I(green) picks out the 1—tuples consisting of elements of D that are green. I(choses) picks out all the pairs (x, y) of elements of D such that x chases y. Similarly, I would map a three—place predicate onto a set of ordered triples, and so on.

Now we can define, in a precise way, what it means for a formula to be true:

- A formula of the form *predicate aYg, arg, . . .*) is true if and only if (I arg), I(nrgz), . . .) e I(predicate).
For example, *chases fdo, felix*) is true when (I(fdo), I(fe/iz)) o I(chases).
- A formula that contains quantifiers or connectives such as , , v, 4, 3 is true if it meets the conditions given by the definitions of the connectives and quantifiers (Table 7.1).

TABLE 7.1 LOGIC SYMBOLS.

Symbol	Read as	Example	Meaning
	not	$\neg P$	P is not true
	and	$P \wedge Q$	P is true and Q is true
$\vee$	or	$P \vee Q$	P is true or Q is true, or both
	implies	$P \rightarrow Q$	if P is true then so is Q (either P is false or Q is true, or both)
$\forall$	for all	$(\forall x) P$	P is true for all values of x
$\exists$	for some	$(\exists x) P$	P is true for at least one value of x
λ	lambda	$(\lambda x) p$	the formula is incomplete and has no truth value until a value is supplied for x

all possible values of x (i.e., true no matter which element of D we assign to x)

Compared to other ways of evaluating logical formulas, model theory has two important advantages. First, it assigns meanings to all parts of every formula, rather than just assigning truth values to complete sentences. Second, model theory works with knowledge bases (models) without making any claims about the real world as a whole. This is important because it corresponds closely to computer manipulation of a database.

Exercise 7.2.1.1

Given the model

$D = \{\text{Fido, Felix, Max}\}$

$I(\text{ludo}) = \{\text{Fido}\}$

$I(\text{c}/\text{iz}) = \{\text{Felix}\}$

$I(\text{mcx}) = \{\text{Max}\}$

$I(\text{mimes}) = \{(\text{Fido}), (\text{Felix})\}$

$I(\text{c}/\text{tnsrs}) = \{(\text{Fido, Felix}), (\text{Max, Fido})\}$

determine whether each of the following formulas is true or false, and show how you obtained each result:

animal fido

animal fido *chases*(*max*, *iz*)

animal(*max*) $\rightarrow$ *animal ifido*

(ix)(*animal*(*x*) $\rightarrow$ *chases*(*x*, *max*))

Exercise 7.2.1.2

Assume that it is true that Fido shows Felix to Max, and that shows is a three—place predicate. Supply $I(\text{shows})$.

7.2.2 Simple Words and Phrases

Table 7.2 shows logical formulas that represent a number of simple English words and phrases, along with a way of representing these formulas in Prolog. As in Chapters 2 and 3, we represent lambda as λ , which is right-associative so that $Y\lambda X$ formula $\lambda Y (X \text{ formula})$

The first thing to note is that names are logical constants ($\text{Max} = \text{max}$), but

common nouns, like adjectives, are predicates ('dog' = $\lambda x \text{dog}(x)$). Being a dog, like being green, is a *property*, not a thing.

This has to do with the distinction between sense and reference. A name can refer to only one individual, so we translate it directly into a logical constant.' But a

'At least for the moment. Montague (1973) and others argue that even names should be treated as denoting sets of properties.

TABLE 7.2 REPRESENTATIONS OF SIMPLE WORDS AND PHRASES.

Type of constituent	Logical representation	As written in Prolog
t	Logical constant	
Proper noun	<i>max</i>	<code>max</code> <code>fido</code>
<i>Max</i> <i>Fido</i>	<i>fido</i>	
Common noun	1-place predicate	
<i>dog</i>	$(Xx)dog(x)$	<code>X' do@ (1)</code>
<i>prof'essor</i>	$(L:x)professor(x)$	<code>I' proIessor (1)</code>
Adjective	1—place predicate	
<i>green</i>	$(Xx)green(x)$	<code>X' green(X)</code>
<i>big</i>	$(Xx)big(x)$	<code>X' big(X)</code>
Noun with adjectives	1—place predicates joined by 'and'	<code>X'' (green(X) , dog(X))</code>
<i>green dog</i>	$(7x)(green(x) \wedge dog(x))$	
Verb phrase	1—place predicate	<code>X'' barked(X)</code> <code>X'' chased (X, fe1 ix)</code>
<i>barked</i> <i>chased Felix</i>	$(7x)barked(x)$ $(7x)chased(x, J'elix)$	
Transitive verb	2—place predicate	<code>Y' X' chased (1, Y)</code>
<i>chased</i>	$(by)(Zx)c/tnsed(z, 1)$	

Copular verb phrase	1—place predicate	
<i>is a dog</i>	$(\exists x)dog(x)$	$X''dog(X)$
<i>is green</i>	$(Xx)green(x)$	$X''green(X)$
Prepositional phrase	1-place predicate	$X'with(X, max)$
<i>with Max</i>	$(ix)with(x, max)$	
Preposition	2-place predicate	
<i>with</i>	$(\lambda x)(\lambda y)with(x, y)$	$Y'X'with(X, Y)$

common noun such as 'dog' can refer to many different individuals, so its translation is the property that these individuals share. The referent of 'dog' in any particular utterance is the value of x that makes $dog(x)$ true.

Second, note that different verbs require different numbers of arguments. The intransitive verb 'barked' translates to a one-place predicate $Xx)barked(x)$. A transitive verb translates to a two-place predicate; a ditransitive verb such as *give* translates to a three-place predicate such as $(Xz)(Yy)Xx)give(x, y, z)$.

These arguments are filled in, step by step, as you progress up from verb to VP

Verb phrase	<i>chases Felix</i>	<i>Xx)chases(x, felix)</i>
Sentence	<i>Fido chases Felix</i>	<i>chases fdo, felix)</i>

We saw this process in action in Chapter 3.

Some sentences correspond to formulas with logical connectives in them:

Fido does not chase Felix *chases fdo, felix)*

Fido and Felix are animals *animal fido)* *animal Felix)*

Note in Table 7.2 that the copula (*is*) is unusual among verbs because it has no semantic representation. To put this another way, *is* X means the same thing as X. The representation for ‘is a dog’ is the same as for ‘dog.’

Last, note that Table 7.2 does not cover the whole of English. Neither does first-order predicate logic. Here are some examples where first—order logic is not sufficient to represent English:

- Predicate with a predicate as argument:

Max has an unusual property.
p Up *unusual (p))*

- Predicate with a whole proposition as an argument:

Fido believes Felix is human.
believes fido, human Felix))

- Context in which reference is blocked:

John is looking for a unicorn.

(It’s not clear how to represent “a unicorn” here. Any formula that contains *(ix) . . . unicorn(x)* is wrong because this unicorn need not exist.)

We will not try to deal with such cases here, but they are all-pervasive; you cannot analyze much natural-language text without running into them. Fortunately, if you confine yourself to texts whose information content can be represented in a computer database, first—order logic or something close to it is usually sufficient.

Exercise 7.2.2.1

Using Tables 7.1 and 7.2, represent each of the following sentences as a logical formula:

Max is angry.

Fido chased Felix.

Felix did not chase Fido.

Either Felix is green, or Fido is blue, or froth.

Fido is a dog and Felix is a cat.

If Max is angry then Fido is angry.

Fido is a green dog.

Fido is either a dog or a cat, but not both.

7.2.3 Semantics of the N^1 Constituent

Now let's do something practical. Recall the syntax rules:

$$\begin{array}{ll} N' & Adj\ N' \\ N' & N \end{array}$$

The N' constituent is a common noun together with zero or more adjectives, but no determiner, such as *big green dog* or */a/ processor*.

Let's implement the semantics of the N^0 . Adjectives and common nouns alike translate to one-place predicates. What we want to do is combine all the predicates in the N' , joining them with ('and'). From

$$\begin{array}{ll} big & \text{--- } (\exists x)big\ x) \\ green & \text{--- } ix)green\ x) \\ dog & \text{--- } ix)dog\ x) \end{array}$$

we want to get:

$$big\ green\ dog \text{--- } ix)\ big\ x)\ green(x)\ dog\ x)).$$

Switching to Prolog notation, we want to combine x "big (X) , X "green (X) , and X "dog (X)

to get $X''(big(X), green(X), dog(X))$. Crucially, we have to ensure that the variables get unified with each other; it would be quite unsatisfactory if we got $(big(X), green(Y), dog(Z))$.

All this is easy to accomplish through arguments on DCG rules. First the lexical entries for particular words:

```
adj(X'big(X)) --> [big].
```

```
adj(X'brown(X)) --> [brown].
```

```
adj(X'little(X)) --> [little].
```

```
adj(X'green(X)) --> [green].
```

```
n(X'dog(X)) --> [dog].
```

```
n(X'cat(X)) -- [cat].
```

Now for the PS rules. The rule $N' \rightarrow N$ is easy to handle, since in this case the semantics of the whole N' is the same as that of the noun:

$n1(X'(P, Q)) \rightarrow adj(X'P), n1(X'Q).$

Notice that this rule is recursive. After combining one adjective with an N' , it combine another adjective with the resulting N' , and so on, as shown in Figure 7.1.

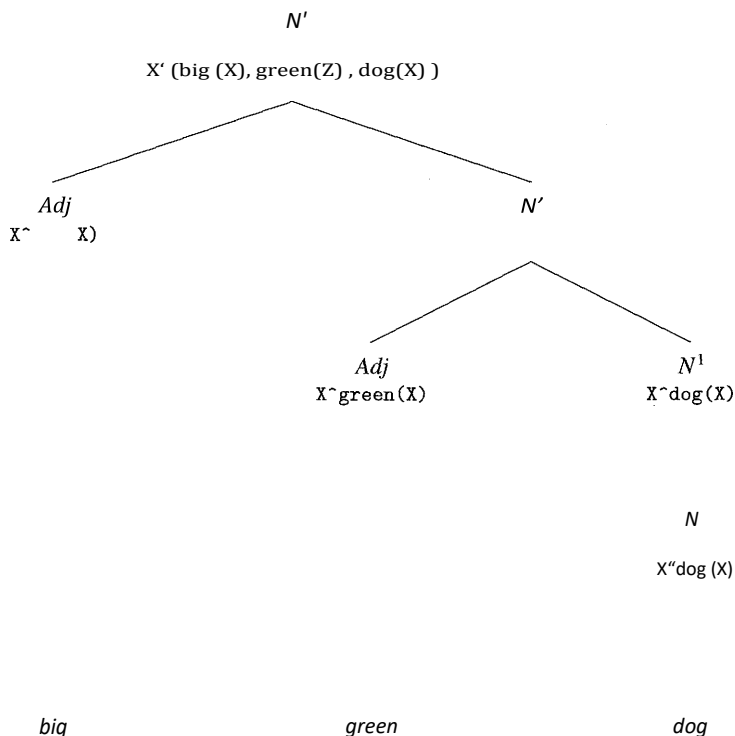


Figure 7.1 Constructing the semantics of an N^0 .

Exercise 7.2.3.1

What would happen if, in place of

$n1(X'(P, Q)) \rightarrow adj(X'P), n1(X'Q).$

the last rule were written as follows?

$$\text{n1}((P, Q)) \rightarrow \text{adj}(P), \text{n1}(Q) .$$

Exercise 7.2.3.2

Using the rules just given, get a small parser working and show that it generates the correct semantics for the phrases *cat*, *big cat*, *big brown dog*, and *big green cat*.

Sec. 7.3 Quantifiers (Determiners)

Exercise 7.2.3.3

Extend this small parser so that:

- It handles NPs whose determiner is *n* or *no*, so that, for example, the semantics of a *big green dog* is the same as that of *big green dog*.
- It handles sentences of the form *Name is NP*, such as *Felix is a big brown cat*, and generates correct semantics, in this case
`(big(felix), brown(felix), cat(felix))`

7.3 QUANTIFIERS (DETERMINERS)

7.3.1 Quantifiers in Language, Logic, and Prolog

Determiners in natural language correspond to quantifiers in formal logic; we shall use the terms *quantifier* and *determiner* almost interchangeably.² Table 7.3 shows some sentences that contain quantifiers and their semantic representations.

The alert reader will notice that the quantifier *3* normally goes with the connective *A*, and *4* with *→*. In our Prolog renditions of the logical formulas, the connectives are implicit. We write at 1 `(X, Coal1, Goal2)` to mean that all values of *X* which satisfy *Coal1* also satisfy *Goal2*. A predicate to test this, in database queries, can be defined as follows:

```
all(_, Goal1, Goal2) :
    \+ (Goal1, \+ Goal2).
```

That's very simple: verify that there is no way to satisfy *Goal1* that cannot be extended (by instantiating more variables) to satisfy *Goal2*. The argument *X* is not actually used; we include it only so that `all` will have the same argument structure as other quantifiers to be defined later on.

But wait a minute—what if there is no way to satisfy Goal1 at all? Then $\text{all}(X, \text{Goal1}, \text{Goal2})$ succeeds, and that's probably not what we want. In logic, $(\forall x (p(x) \supset q(x)))$ is true in the situation where $p(x)$ is always false. But in natural language, we don't want to claim that *all unicorns are green* is true when there are no unicorns in the knowledge base. Accordingly, we should modify `all` to test that there is indeed at least one solution to Goal1, and then cut to prevent pointless backtracking:

```
% all(-X,+Goal1,+Goal2)

% Succeeds if all values of X that satisfy Goal1
% also satisfy Goal2.
```

²Some authors say that a quantifier consists of a determiner plus its restrictor; that is, a determiner by itself is not a complete quantifier, but only an ingredient for making one. On this view, for example, 'most

Goal1,

The definition of `some` is even simpler, because all we have to do is find *one* of `Goal1` and `Goal2`. Again, we use a cut to prevent pointless backtracking:

```
% some(-X,+Goal1,+Goal2)

% Succeeds if there is a value of X that
% satisfies Goal1 and Goal2.

some(_,Goal1,Goal2) :-
    Goal1,

    Goal2,
```

Exercise 7.3.1.1

Referring to Table 7.3 as needed, express each of the following sentences as a logical formula, and as the Prolog representation of that formula.

All unicorns are animals.

One or more unicorns are purple.

A cat chased Fido.

keep cat chased Fido.

keep cat is an animal.

keep cat chased a dog.

Exercise 7.3.1.2

When `all / 3` succeeds, does it instantiate its first argument? If so, explain how.

Exercise 7.3.1.3

What is the purpose of the cut in `some / 3`?

Exercise 7.3.1.4

Given the knowledge base

```
dog(fido).  
cat(felix).  
cat(leo).  
animal(leo).  
animal(felix).  
animal(fido).
```

TABLE 7.3 QUANTIFIERS DETERMINE THE OVERALL SEMANTIC STRUCTURE OF THE SENTENCE.

Sentence	Representations
<i>Fido barked.</i>	$\text{barked}(\text{fido})$ $\text{barked}(\text{lido})$
<i>A dog barked.</i>	$(\exists x)(\text{dog}(x) \text{ barked}(x))$ $\text{some}(X, \text{dog}(X), \text{barked}(X))$
<i>Every dog barked.</i>	$(\forall x)(\text{dog}(x) \rightarrow \text{barked}(x))$ $\text{all}(X, \text{dog}(X), \text{barked}(X))$
<i>Fido chased a cat.</i>	$(\exists s)(\text{cat}(\text{v}) \text{ chased fido}, x))$ $\text{some}(X, \text{cat}(X), \text{chased}(\text{fido}, X))$
<i>Fido chased every cat.</i>	$(\forall x)(\text{cat}(x) \rightarrow \text{chased fido}, x))$ $\text{all}(X, \text{cat}(X), \text{chased}(\text{fido}, X))$
<i>A dog chased a cat.</i>	$(\exists x)(\text{dog}(x) \quad (\exists y)(\text{cat}(y) \quad \text{chased}(x, y)))$ $\text{some}(X, \text{dog}(X), \text{some}(Y, \text{cat}(Y), \text{chased}(X, Y)))$
<i>A dog chased every cat.</i>	$(\exists x)(\text{dog}(x) \quad (\forall y)(\text{cat}(y) \quad \text{chased}(x, y)))$ $\text{some}(X, \text{dog}(X), \text{all}(Y, \text{cat}(Y), \text{chased}(X, Y)))$
<i>Every dog chased a cat.</i>	$(\forall x)(\text{dog}(x) \rightarrow (\exists y)(\text{cat}(y) \quad \text{chased}(x, y)))$ $\text{all}(X, \text{dog}(X), \text{some}(Y, \text{cat}(Y), \text{chased}(X, Y)))$ $(\forall x)(\text{dog}(x) \rightarrow (\forall y)(\text{cat}(y) \rightarrow \text{chased}(x, y)))$
<i>Every dog chased every cat.</i>	$\text{all}(X, \text{dog}(X), \text{all}(Y, \text{cat}(Y), \text{chased}(X, Y)))$

predict (without using the computer) the outcomes of the queries:

?- some (X, dog (X) , animal (X)) .

°- all (X, cat (X) , animal (X)) .

?- all (X, animal (X) , cat (X)) .

Then use the computer to confirm your results.

7.3.2 Restrictor and Scope

We noted already that 3 is somehow associated with and 9 with m. In fact, in Prolog, we left out the latter connective and treated each quantifier as a relationship between a quantified variable and two Prolog goals.

Let's pursue this idea further. Consider the formulas

$(\exists x)(\text{cat}(x) \rightarrow \text{animal}(x))$ 'At least one cat is an animal.'

These can be written another way. Let $(\exists x : cat(x))$ mean 'For all z such that $cat(z)$ in cat.' Then, using this notation, we can write:

$$\begin{aligned} (\exists x : cat(x)) animal(x) & \text{ 'At least one cat is an animal.'} \\ (\forall x : cat(x)) animal(x) & \text{ 'Every cat is an animal.'} \end{aligned}$$

Notice that the connectives $\wedge$ and $\rightarrow$ have disappeared.

In this new notation, $cat(x)$ is called the *restrictor* (of RANGE) Of the quantifier $\exists x$ or $\forall x$; it restricts the set of values of z that the quantifier can pick out. The rest the formula, here $animal(x)$, is the *scope* of the quantifier; it is the proposition that is supposed to be true for the appropriate values of x .

From this perspective we can look at each quantifier as a relation between quantified variable, scope, and restrictor:

- $(\exists x)$ means that *at least one* value of x that makes the restrictor true will also make the scope true;
- $(\forall x)$ means that *every* value of x that makes the restrictor true will also make the scope true,

More precisely, for any quantified variable z , the quantifier is a relation between the set of values of x that satisfy the restrictor, and the set of values of a that satisfy both the restrictor and the scope.

It's easy to invent other quantifiers, such as:

- $(\exists! x)$, meaning that exactly two of the values of x that satisfy the restrictor also satisfy the scope;
- $(\text{most } x)$, meaning that more than half of the values of x that satisfy the restrictor also satisfy the scope;

or whatever you like. These are called **GENERALIZED QUANTIFIERS** and are essential for analyzing natural language.

We can represent generalized quantifiers in Prolog with three-place predicates such as `is_a(X, Y, Z)`, `is_a(X, Y, Z)`, `is_a(X, Y, Z)`, and so forth. Now it is obvious why X has to be an argument (even though some and all didn't use it): quantifiers

like *owe* and *more* have to count the values of x that satisfy the goals, and they have to make sure they are indeed counting values of X rather than values of some other variable.

Exercise 7.3.2.1

Rewrite each of your formulas from Exercise 7.3.1.1 in restrictor-scope format

Exercise 7.3.2.2

Consider the following ledger Use:

```
dog(fido).  
cat(felix).
```

```
cat(leo) .
chases(fido,felix) .
chases(fido,leo) .
```

How many solutions are there to the query ? – `dog(X) , cut(Y) , ches en(X,Y) .` ? How many dogs are cat-chasers? Why aren't these the same number, and what does this tell us about the right way to implement generalized quantifiers?

Exercise 7.3.2.3

Define the generalized quantifier `two` in Prolog using `select` of (see section 7.4.2.)

Exercise 7.3.2.4

In ordinary English, does *two* mean 'exactly 2' or does it mean '2 or more' ? Discuss and cite evidence.

7.3.3 Structural Importance of Determiners

Determiners affect more of the *semantic* structure than the *syntactic* structure would suggest.

Consider for example *Fido chases every cat*. On the syntactic level, *every* modifies *cat*. You might therefore expect that on the semantic level, when *every* is converted into an `ml l(.,.,.)` structure, only *cat* will be inside it.

But such an expectation would be wrong. On the semantic level, *every* has scope over the entire sentence, even though syntactically it only modifies *cat*. Figure 7.2 shows part of the process by which the semantic structure is built up.

Or consider *Every dog chases a cat*, which we represent as:

```
all(X,dog(X),someY,cat(Y),chases(X,Y))
```

Here the constituent *every dog* gives rise to an `at 1` structure that contains not only the representation of the NP (where *every* occurs), but also the representation of the VP.

Does this mean that our enterprise of building semantic structures constituent-

by- constituent is doomed? No; it means only that some careful use of lambda expressions is required.

Exercise 7.3.3.1

Draw diagrams like Figure 7.2 for the sentences:

Every cat chased Fido.

Some dog chased every cat.

7.3.4 Building Quantified Structures

Now for the implementation. Despite their complexity, the semantic structures that we need can still be built through unification of arguments in DCG rules.

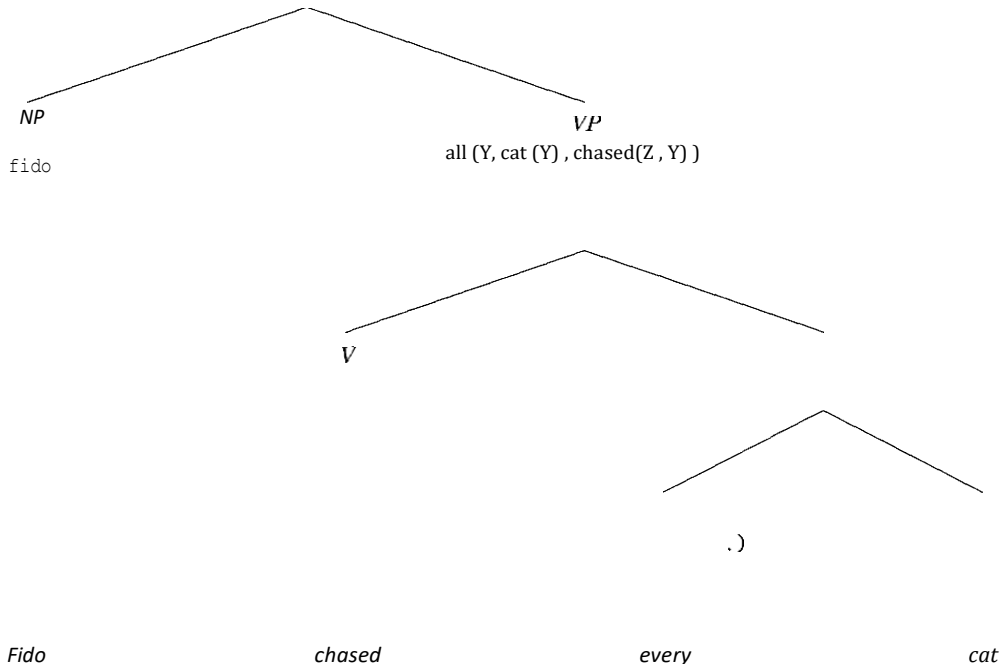


Figure 7.2 The quantifier *all* dominates the semantic representation, even though syntactically it belongs only to the last NP. (This structure is incomplete; all lambdas are left out.)

Our representation of verbs and common nouns will be the same as before. For brevity we will omit the N' node. This gives us some lexical entries:

$n(X' \text{ dog}(X)) \rightarrow [\text{dog}]$.

$n(X' \text{ cat}(X)) \rightarrow [\text{cat}]$.

$v(X'' \text{ meowed}(X)) \rightarrow [\text{meowed}]$.

$v(Y' X' \text{ chased}(X, Y)) \rightarrow [\text{chased}]$.

$v(Y' X' \text{ saw}(X, Y)) \rightarrow [\text{saw}]$.

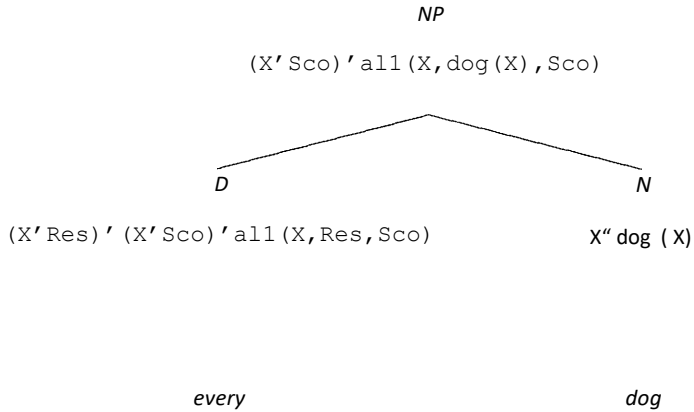
The key question at this point is how to represent determiners. For our purposes, a determiner is something that takes a scope and a restrictor, and puts them together. So the semantic representation of every determiner will have the form

$(X \text{ "Res}) \text{ " } (X \text{ " Sao)}$ Formula —that is, “Give me a restrictor and a scope and I’ll make a formula out of them.” The variable X is explicit here so that the corresponding variables in different terms will be unified. Thus we get the lexical entries:

$d((X \text{ ' Res}) \text{ ' } (X \text{ ' Sco}) \text{ ' all}(X, \text{Res}, \text{Sco})) \rightarrow [\text{every}].$

$d((X \text{ ' Res}) \text{ ' } (X \text{ ' Sco}) \text{ ' some}(X, \text{Res}, \text{Sco})) \rightarrow [a]; [\text{some}].$

Now that we know what a determiner is, what's an NP? The same thing as a determiner, except that the restrictor has been supplied, as in the structure:



The D and NP are alike except that the NP is only awaiting the scope, not the restrictor. The grammar rule for NP is

$\text{np}((X' \text{ Sco})' \text{ Pred}) \rightarrow \text{d}((X' \text{ Res})' (X' \text{ Sco})' \text{ Pred}), \text{n}(X' \text{ Res}).$

or, more concisely,

$\text{np}(\text{Sem}) \rightarrow \text{d}((X' \text{ Res})' \text{ Sem}), \text{n}(X' \text{ Res}).$

There are two VP rules. If the verb has no object, then the semantics of the VP is the same as that of the V:

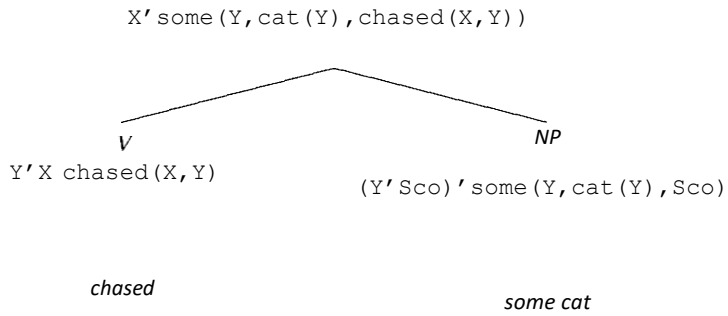
$\text{vp}(\text{Sem}) \rightarrow \text{v}(\text{Sem}).$

That gives us $[\text{VP meowed}] \rightarrow X''\text{meowed}(X)$ and the like.

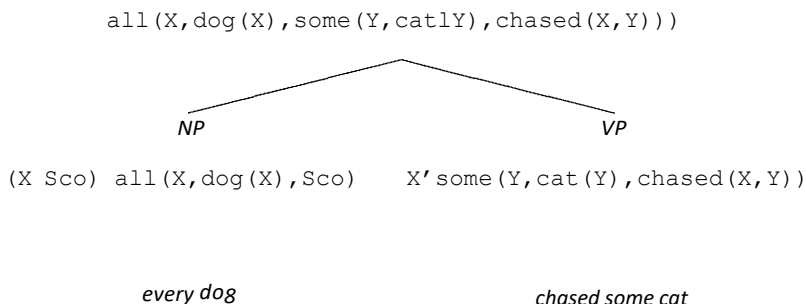
But if there is an object NP within the VP, the verb becomes the scope of that NP, thus:

$\text{vp}(X' \text{ Pred}) \rightarrow \text{v}(Y' X' \text{ Sco}), \text{np}((Y' \text{ Sco})' \text{ Pred}).$

This accounts for structures such as:



Finally, the S rule takes the NP (which is waiting for a scope) and the VP (which is waiting for an argument), and puts them together:



Notice how things have changed: in Chapters 3 and 5 we treated the subject as an argument of the VP, and now we're doing it the other way around, so that determiners (and NPs containing them) will always have control of the overall semantic structure.

That's enough to handle many common sentence structures. Figure 7.3 shows, in detail, how the structure of *Every dog chased some cat* is built up.

One last problem remains. We can no longer represent proper names as constants; instead, they have to be structures that accept a VP as scope. Instead of

```
np(fido) --> [fido].
```

we have to write

```
np((fido'Sco)'Sco) --> [fido].
```

Here `fido'Sco` "Sao is an expression that receives `X'Sco` from the verb phrase, instantiates the lambda variable to `fido`, and returns `Sno` with no other changes.

Exercise 7.3.4.1

Get a parser working that uses the grammar rules introduced in this section, plus other rules as necessary, in order to generate correct semantic representations for the sentences:

Fido barked.

A dog barked.

Every dog barked.

Fido chased Felix.

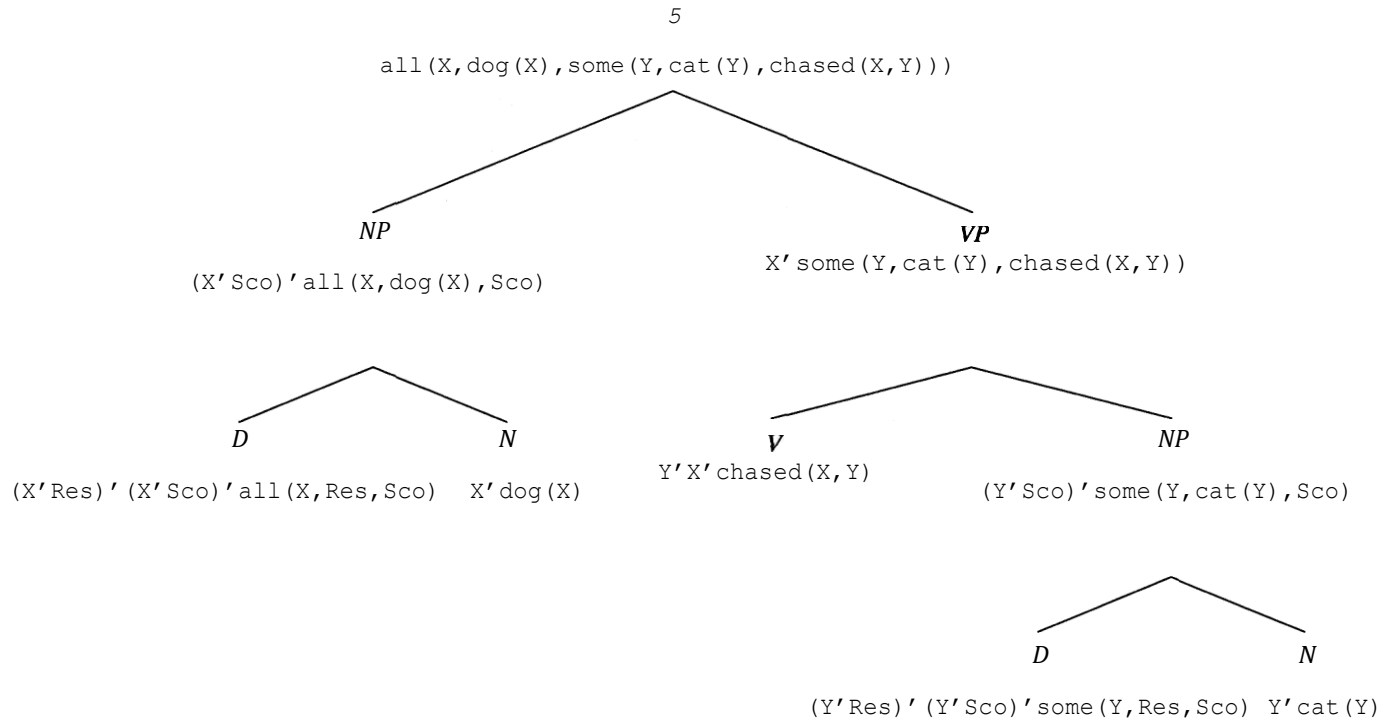
A dog chased Felix.

Felix chased a cat.

Every dog chased Felix.

*Felix chased every cat. A
dog chased every cat.*

Every dog chased a cat.



every

dog

chased

some

cat

Figure 7d Building semantic structure, in detail.

The alert reader will have noticed that *A dog chased every cat* is ambiguous. Here are its two readings:

A dog chased every cat —

- (1) $\text{some}(X, \text{dog}(X), \lambda Y. \text{cat}(Y), \text{chased}(X, Y))$
‘There is a dog that chased all cats’
- (2) $\lambda Y. \text{cat}(Y), \text{some}(X, \text{dog}(X), \text{chased}(X, Y))$
‘Each cat was chased by some dog (not necessarily the same dog)’

The first of these is what our rules so far generate. The second can be derived from the first by a transformation known as QUANTIFIER RAISING:

$\text{some}(X, \text{dog}(X), \lambda Y. \text{cat}(Y), \text{chased}(X, Y))$

$\lambda Y. \text{cat}(Y), \text{some}(X, \text{dog}(X), \text{chased}(X, Y))$

or, more generally:

$Q_1(V_1, R_1, Q_2(V_2, R_2, S_2))$

(QUANTIFIER RAISING FROM SCOPE)

$Q_2(V_2, R_2, Q_1(V_1, R_1, S_2))$

where Q_1 and Q_2 stand for the quantifiers³ and R_2 does not contain any variable other than V_2 .

It’s also possible to raise a quantifier from the restrictor, like this:

Someone who sees every dog laughs. —

- (1) $\text{some}(X, \lambda Y. \text{dog}(Y), \text{sees}(X, Y), \text{laughs}(X))$
‘There is someone who sees all the dogs and laughs’
- (2) $\lambda Y. \text{dog}(Y), \text{some}(X, \text{sees}(X, Y), \text{laughs}(X))$
‘For each dog, there is someone who sees it and

laughs’ This time the general schema is:

$Q_1(V_1, Q_2(V_2, R_2, S_2), S_1)$

$Q_2(V_2, R_2, Q_1(V_1, R_1, S_2))$

(QUANTIFIER RAISING FROM RESTRICTOR)

'Of course real Prolog does not allow a variable in this position.

Actually these two schemas are instances of a more general pattern. In fact almost *any* quantifier anywhere within the scope or restrictor—no matter how deep down—can be raised to have scope over the entire sentence. Here's how to do it:

- Pick the quantified structure to be raised; call it $Q\ I\ V, R, S$
- Replace $Q\ (V, R, S)$ with S . Call the resulting formula F .
- The result of raising is then $Q\ (V, R, F)$

That describes how to raise one quantifier from anywhere in the sentence. In real life, more than one quantifier can be moved, because quantifier raising is recursive. There is considerable debate as to how far the recursion should be allowed to go.

In principle, a sentence with n quantifiers could have $n!$ (n -factorial) readings. To see that this is so, consider that you can choose any of the n quantifiers to be raised to the topmost position. Having done this, you can then raise any of the $u - 1$ quantifiers not yet raised; then any of the $n - 2$ remaining, and so on, yielding $n \times (n - 1) \times (u - 2) \times \dots$ alternatives.

In practice, we don't get this many, for several reasons. Some readings are blocked because they leave variables unbound (that is, they put variables outside the scope or restrictor of the quantifiers that bind them). Other readings are logically equivalent (for example, raising one *all* past another *all* has no effect on the truth conditions of the sentence). Still others are blocked by structural principles. For example, Hobbs and Shieber (1987) point out that in English, it is not permissible to take a quantifier from outside an NP and put it between two quantifiers that originated in that NP.

Further, there are differences in the ease with which various quantifiers can be raised. Specifically, *each* almost always raises; *some*, *all*, and *every* can raise but need not do so; *any* tends not to raise; and numerals generally do not raise.

The *especially* alert reader will now notice that quantifier raising is what we've been doing all along. The whole point of Figures 7.2 and 7.3 was that quantifiers get raised out of individual NPs so that they have scope over the complete sentence. So it makes sense to build an initial semantic representation with the quantifiers still in the NPs, and then require that they be raised far enough to give well-formed logical

formulas, with further raising then being optional.

That is in fact now the standard approach to quantifier scoping (Cooper 1983, Hornstein 1984, Chierchia and McConnell–Ginet 1990 chapter 3). Hobbs and Shieber (1987) give an algorithm that raises quantifiers from NP positions, generating all and only the scopings that are structurally possible in English.

Exercise 7.3.5.1

Give formulas for the two readings of *Every man loves a woman*.

Exercise 7.3.5.2

Explain why quantifier raising has no effect on the truth conditions of *Every dog chased*

Everyone who hates a dog loves two cats.

as a formula. Derive two more formulas from it (by raising from the scope and from restrictor respectively) and explain the meaning of the sentence that corresponds to each of your three formulas.

Exercise 7.3.5.4

Implement quantifier raising from scope and from restrictor. That is, implement a predicate `raise_quantifier / 2` which, when given a formula, will perform raising from the scope or from the restrictor, whichever is possible, and will also have a solution in which no raising is performed. For example:

```
?- raise_quantifier(some(X,dog(X),all(Y,cat(Y),chased(X,Y))),What).
What = all(Y,cat(Y),some(X,dog(X),chased(X,Y)))
What = some(X,dog(X),all(Y,cat(Y),chased(X,Y)),What).
```

When given the formula from the previous exercise, `raise_quantifier` should get all three readings. You need not check whether variables are left unbound (in these examples they won't be).

Exercise 7.3.5.5

Define a predicate `inLerpreL/2` that parses a sentence using your parser, then optionally applies quantifier raising, so that if you give it `[every, dog, eta oed, a, cat]` you get two alternative formulas.

7.4 QUESTION ANSWERING

7.4.1 Simple Yes/No Questions

We have now implemented enough semantics to be able to answer, from a knowledge base, plain-English questions such as:

Did Fido chase Felix?

Did every cat meow?

Did every dog chase a cat?

All we need to do is change the grammar rules so that questions can be parsed. To do this, we replace the S rule

$s(\text{Sem}) \rightarrow np((X' \text{ Sco})' \text{ Sem}), vp(X' \text{ Sco}) .$

with one appropriate for
questions,

$s(\text{Sem}) \rightarrow [\text{did}], \text{np}((\text{X}' \text{Sco})' \text{Sem}), \text{vp}(\text{X}' \text{Sco}) .$

and we add the “plain” forms of the verbs (without final *-ed*),
thus:

$v(\text{X}' \text{meowed}(\text{X})) \rightarrow [\text{meowed}]; [\text{meow}] .$

$v(\text{Y}' \text{X}' \text{chased}(\text{X}, \text{Y})) \rightarrow [\text{chased}]; [\text{chase}] .$

$v(\text{Y}' \text{X}' \text{saw}(\text{X}, \text{Y})) \rightarrow [\text{saw}]; [\text{see}] .$

Now *Did every dog chase a cat?* translates directly into the query

$\text{all}(\text{X}, \text{dog}(\text{X}), \text{some}(\text{Y}, \text{cat}(\text{Y}), \text{chased}(\text{X}, \text{Y})) .$

which can be answered from a suitable knowledge base.

Exercise 7.4.1.1

Using the grammar rules presented so far, the definitions of some and at 1, and the knowledge base

```
cat(felix).      cat(leo).
dog(fido).       dog(bucephalus).
saw(felix,fido).
saw(leo,felix).
chased(fido,felix).
chased(bucephalus,leo).
```

write a Prolog program that will accept and answer the questions

Did Leo see Felix?

Did every cat see a dog?

Did a cat see a dog?

Did every dog chase a cat?

and others of similar form. You need not handle scope ambiguities.

7.4.2 Getting a List of Solutions

To go further and answer questions that contain *which* or how many, we will need a way to get a list of solutions to a Prolog query. Two approaches are possible.

The built-in predicate `solve/3` returns a list of all solutions to a query. More precisely,

is sorted in alphabetical order and duplicates are removed. For example:

```
dog(fido) .  
dog(bucephalus) .  
  
?- setof(X,dog(X),L) .  
L = [bucephalus,fido]
```

That's *almost* what we need. The problem is that if Coal contains a variable other than X, `sed o I` will return only the solutions in which that variable has one particular value. On backtracking, `sed of` will then let that variable have another value, and so on. With the knowledge base

```
f(a,c) .  
f(b,c) .  
f(c,d) .  
f(d,d) .
```

we get:

```
?- setof(X,f(X,Y),L) .  
Y = c      L = [a,b] ;  
Y = d      L = [c,d] ;
```

What we want is to get all the values of X into a single list. Otherwise, there are some queries we can't answer, such as how many dogs chase any cat (not necessarily the same cat).

Fortunately `set of` provides a way to work around this.' If we write

```
?- setof(X,Term'Goal,L) .           % here    does not denote lambda
```

then `sed o I` works as before, except that all the variables in Term are allowed to take on all possible values, like this:

```
°- setof(X,Y'f(X,Y),L) .  
L = [a,b,c,d]
```

(assuming the same knowledge base as before).

What we want, of course, is this kind of treatment for all the variables in Goal, so we simply write Coal “Noa1. We'll encapsulate this trick by defining the predicate `solutions/3`:

⁴At least in ALS, Arity, and Quintus (UNIX) Prolog, and in the draft ISO standard; not in LPA Prolog

3.10 (Quintus Prolog for MS—DOS).

```
% solutions(-X,+Goal,-List)
% Returns, in List, all the values of X that satisfy Goal.
% Free variables in Goal are allowed to take all possible values.
% List contains no duplicates.

solutions(X,Goal,List)
    setof(X,Goal'Goal,List). % here does not denote lambda
```

A few Prologs either do not have `setof`, or do not accept the Goal “Goal trick. These Prologs have, instead, a built-in predicate `findall/3` that works like `setof`, but allows the free variables to take on all values, thus:

```
°- findall(X,f(X,Y),L).
L = [a,b,c,d]
```

The trouble with `findall` is that it doesn’t remove duplicates. If we’re counting the cat—chasing dogs, we don’t want to count Fido twice merely because he chases two cats. So we’ll need to define `solutions` as:

```
solutions(X,Goal,List)
    findall(X,Goal,L),
    remove_duplicates(L,List).
```

You will have to define `remove_duplicates` also.

Exercise 7.4.2.1

Get `solutions/3` working on your computer. Using a small knowledge base, verify that it works correctly.

Exercise 7.4.2.2

Using solutions / 3, define the following generalized quantifiers:

- $\text{two}(X, \text{Res}, \text{Sco})$, true if there are exactly two values of X that satisfy Res and also satisfy Sco ;
- $\text{three}(X, \text{Res}, \text{Sco})$ and $\text{four}(X, \text{Res}, \text{Sco})$, analogous to two;
- $\text{mos}(X, \text{Res}, \text{Sco})$, true if more than half of the values of X that satisfy Res also satisfy Sco .

Be sure you compare the number of values that satisfy Res to the number of values that satisfy *both* Res and Sco (not just Sco by itself). If there are three dogs and two green objects (not dogs) in the knowledge base, you would not want to conclude that two dogs are green.

7.4.3 *Who/What/Which* Questions

Now we can tackle questions that contain the “wH-words” *who*, *what*, *which*, and *how*

here.

We can handle *which* and *how many* like quantifiers, except that when they occur

in a query, the Prolog system will display the values on the screen rather than just testing the truth of a statement:

```
which(X, Res, Sco) :-
    solutions(X, (Res, Sco), L),
    write(L),

    nl.

how_many(X, Res, Sco) -
    solutions1X, (Res, Sco), L,
    length(L, N),

    write(N),
    nl.
```

The main syntactic peculiarity of wH-words is that they always occur at the beginning of the sentence. Of course, sometimes they would have been there anyway; structures like

Which ant saw a dog?

How many cats saw a dog?

are no problem. In other cases, however, the wH—word moves to sentence—initial position and takes the rest of its NP with it. Instead of

Fido chased which cat?

we normally get

Which cat did Fido chase ?

where, as in Chapter 3, denotes a missing NP, The moved NP can be associated with its original position using the same techniques as in Chapter 3.

WH-words are also subject to scope ambiguities. In English, IVmc/t *dogs chased a cat?* can mean either ‘Which dogs chased one particular cat?’ or ‘Which dogs chased any cat (we don’t care which one)?’ These require the same techniques, and present the same puzzles, as the ambiguities of *all* and *some* already noted.

Exercise 7.4.3.1

Extend your parser from Exercise 7.4.1.1 to answer queries such as:

Which cat saw Fido?

How many dogs chased some cat?

Fido chased how many cats?

You need not deal with scope ambiguities, nor with plurals whose determiner is anything other than *which* or *how many*.

Exercise 7.4.3.2 (small project)

Extend your parser so that it also handles wH-movement and will answer queries such as:

How many cats did Fido chase?

Which cats did every dog see?

Again, you need not handle scope ambiguities, nor plurals with determiners other than *which* or *how many*.

7.5 FROM FORMULA TO KNOWLEDGE BASE

7.5.1 Discourse Referents

When we move from question answering to actually building a knowledge base from English input, a problem immediately arises. Consider how to render the sentence

Max owns a dog.

into Prolog. A first guess would be to use the two facts:

```
dog(X) .                % wrong!
```

```
owns(max,X) .
```

That's not right; in Prolog, these facts say 'Anything is a dog' and 'Max owns anything.' They would succeed with queries as bizarre as `?- dog(qwertyuiop) .` and `'?- owns(max,new_york) .`

Since the dog doesn't have a name, we have to give it one. That is, we have to

recognize the dog as a DISCOURSE REFERENT (a thing that can now be talked about)' and give it a unique name (a DISCOURSE **MARKER**), such as x_{123} or $x_{(123)}$. Then what we want to say is:

$\text{dog}(x_{(123)})$.

$\text{owns}(\text{max}, x_{(123)})$.

We will also need some kind of TABLE OF IDENTITY So that if we ever find out this dog's name, or find out that it is identical to some other dog with a different discourse marker, we can keep track of it properly.

An even bigger problem arises with statements like *Every farmer owns a dog*. What we want is something like this,

$\text{dog}(Y) :- \text{farmer}(X).$ % wrong!

$\text{owns}(X, Y) :- \text{farmer}(X).$

'See Karttunen (1969), an account still well worth reading. In this section I follow Covington, Nute,

name be a structure that has the farmer's name in it, like this:

```
dog(x(124,X))      :- farmer(X) .
```

```
owns(X,x(124,X))  :- farmer(X) .
```

Then if Max is a farmer, his dog is temporarily known as $x(124, \text{max})$; Bill's dog $x(124, \text{bi11})$; and so on. Each dog has a unique name. This is a form of TION (Skolem 1920), the replacement of $\mathcal{L}$ -quantified variables with terms whose depend on the values of all the other variables. In effect, we are changing

$$\exists x (\text{farmer}(x) \wedge \exists y (\text{dog}(y) \wedge \text{owns}(x, y)))$$

into

$$\exists x (\text{farmer}(x) \wedge \neg r(\text{dog}(f(x)) \wedge \text{owns}(x, f(x))))$$

where f is a function that maps each farmer onto (a name for) the appropriate dog.

Exercise 7.5.1.1

Define a procedure `general_epmarker/1` which will create new discourse markers by instantiating its argument to a different discourse marker every time it is called, thus:

```
?- generate_marker(Nhat) .
```

```
What - x(1)
```

```
?- generate_marker(What) .
```

```
What - x(2)
```

```
?- generate_marker(What) .
```

```
What - x(3)
```

and so on. (Hint: Use `assert` and `retract`.)

Exercise 7.5.1.2

Translate into Prolog facts and rules, by hand, using discourse markers where necessary:

A dog barked.

Felix chased a dog. Felix

chased every dog. A dog

chased a cat.

Exercise 7.5.1.3 (small project)

Write a program that will take the formulas

```
some (X, dog (X) , barked (X) )  
some (X, dog (X) , chased (felix, X) )  
all (X, dog (X) , chased (felix, X) )  
some (X, dog (X) , some (Y, cat (Y) , chased (X, Y) ) )
```

(which are, of course, representations of the sentences in the previous exercise) and translate each of them into one or more Prolog clauses using discourse markers wherever appropriate.

7.5.2 Anaphora

ANAPHORA is the use of pronouns (ANAPHORS) to refer to people, places, or things

previously mentioned. For example:

*Max; photographed himself.
Then he; photographed Sharon j
and she j photographed him;.*

The subscripts *i, j, ...* identify words that are coREFERENTIAL (refer to the same person). In order for an anaphor to be understood, it must be matched up with the appropriate pre—existing discourse referent. This is called RESOLVING the anaphoric reference⁶ and is often done by looking for the ANTECEDENT Of the anaphor, i.e., the previous mention of the thing that the anaphor refers to. Anaphora resolution is still an area of ongoing research, but several important principles have emerged.

First, *anaphors stand for discourse referents, not for words or phrases*. Consider the example:

Max found a trail and followed it.

Clearly this means that Max found a trail and then followed the same trail. But if it were merely a substitute for the words a *trail*, then

Max found a trail and followed a trail.

would mean the same thing, which it doesn't (in the latter sentence the two trails need not be the same). Evidently, then, an anaphor stands for the same referent, not merely the same words, as its antecedent.

Occasionally the antecedent of the anaphor is something that has not been mentioned, but has been brought to the hearer's attention some other way. This is called **PRAGMATIC ANAPHORA**. For an example, imagine hearing a loud noise and asking someone, "What one if?" The antecedent of *it* is the noise, which has not been mentioned.

Second, *the antecedent almost always precedes the anaphor*. This is simple but important. The obvious way to search for the antecedent of an anaphor is to start with the most recently introduced discourse referent, and search backward until a suitable antecedent is found.

Occasionally the anaphor and antecedent are in reverse order; that is, the anaphor comes first. This is called **BACKWARDS ANAPHORA** or **CATAPHORA** and the classic example is:

Near him, John, saw a snake.

with the pronoun more deeply embedded (farther from the S node at the top), as by Ross (1967) and many others since (see Carden 1986).

Third, *the gender and number of the anaphor restrict the set of possible antecedents*. In English, we use *he* for singular males, *she* for singular females, *it* for singular inanimate objects, and *they* for plurals.⁷

This suggests a general algorithm for finding antecedents:

- Keep a list (or a series of Prolog facts) listing all the discourse referents, newest first, and tagging each of them as masculine, feminine, or inanimate, and as singular or plural.
- Upon finding an anaphor, search through the discourse referents to find the most recent antecedent with appropriate gender and number.

And in fact this strategy works well; Allen (1987:339–354) explores it at some length. Hobbs (1978) found that it is not usually necessary to search back very far, because 98% of all antecedents are within the current or the previous sentence.

There's more. *The form of the anaphor indicates whether the antecedent is in the same sentence*. In English, intrasentential anaphors (REFLEXIVES) end in *-self*. For example:

John; saw himself.

John; saw himj. (j i)

Here we know that *him* in the second sentence cannot be coreferential with *John* because it does not end in *-self*.

Actually, “in the same sentence” is not quite the right criterion; the exact syntactic criteria are more complicated, and are not fully understood. Note the contrast between:

John; baked a cake for himself. (not him;)

John; saw a snake near him; himself.

The problem of formulating the exact conditions for the use of reflexives has been an important stimulus for research in generative grammar (Chomsky 1982:218 ff., 288 ff.).

Finally, *semantics and real-world knowledge can help choose between possible antecedents*, as in Jespersen's macabre example:

If the baby; does not thrive on raw milk j, boil it j.

Here you have to know that milk can be boiled and babies can't.

⁷ We will get to plural discourse referents in the next section.

'Jespersen (1954:143), cited by Hobbs (1978).

Computational linguists who are daunted by the challenges of anaphora can take solace in the fact that native speakers have problems too, especially when expressing themselves in writing. Unclear or misleading antecedents are a common problem in poorly written English.

Exercise 7.5.2.1

Consider the following short text:

Without considering whether he would offend vegetable growers, the president said he hated broccoli and didn't have to eat it if he didn't want to. And sure enough, they were offended. As a protest, they sent him a huge amount of it.

- (a) Use subscripts to indicate the coreferential nouns and pronouns.
- (b) Point out an instance of cataphora.
- (c) What kinds of information do you rely on when identifying the antecedent of each anaphor? What kinds of indicators mentioned in the text are not necessary here?

Exercise 7.5.2.2

Consider now the much simpler text:

*Cathy photographed Fred.
Then she photographed
herself.*

Finally Fred looked at Sharon and she photographed him.

which goes into formulas as:

```

photographed(cathy,fred) .
photographed(she,herself) .
looked_at(fred,sharon) .
photographed(she,him) .

```

Define a procedure `resolve_anaphors/2` that will accept this series of formulas (in a Prolog list) and will replace all the anaphors with the names of their most likely referents, thus:

```

?- resolve_anaphors([photographed(cathy,fred),
                    photographed(she,herself)],What) .

```

```

What = photographed(cathy,fred),photographed(cathy,cathy) ]

```

(and likewise for the complete list, and for other similar lists).

As real-world knowledge, your procedure can assume that Cathy and Sharon are female and Fred is male. You can further assume that each formula corresponds to a single sentence, and that all anaphors refer to individuals that have been named (so that there is

Although we've analyzed lots of determiners, we still haven't said anything about What does an NP like *the cat* really mean?

The classic analysis, due to Bertrand Russell (1905), is that *the* is a quantifier (written $\exists!$ or ι) which means 'there is exactly one.' On this analysis, *The king is bald* corresponds to the formula

$$(\exists!x : \text{king } x) \text{bald } x)$$

and means 'There is exactly one value of x which satisfies $\text{king}(x)$ and also $\text{bald}(x)$.'

But Russell's analysis captures only part of the picture. Quite often, definite NPs (NPs with *the*) refer to discourse referents already mentioned, thus:

A dog; barked and a cat howled.
Then the dog; chased the cat
away.

If the second sentence had said *A dog chased a cat away* it would have suggested that the second dog and cat are not the same as the first ones. Using *the* makes it clear that the dog and the cat are the ones already mentioned. It is as if *the dog* were an anaphor that can only refer to dogs.

It turns out that treating definite NPs as anaphors is a good idea, with the proviso that pragmatic anaphora is common, and that the antecedent is often in the hearer's back-ground knowledge (or assumed background knowledge) rather than in his or her immediate awareness. I can say *the king of Lesotho* without previously having mentioned him; you will react to this by assuming (if you did not know already) that Lesotho does indeed have a king. That is, you will ACCOMMODATE to my pragmatic anaphora by introducing a discourse referent with appropriate properties. Russell's analysis of f6c does a good job of characterizing the effect of *the* in just this special case where no antecedent is available.

Exercise 7.5.3.1

Modify the sentences from the previous exercise so that it will also resolve the referents of NPs bound by *the*, treating them as anaphors whose referents must satisfy a particular predicate. This time, use the text:

Henry III knighted Robin Hood.
Then Friar Tuck petitioned the king
and the king knighted the friar too.

The formulas that correspond to the sentences are

```
knighted(henry_iii,robin_hood).  
the(X,king(X),petitioned(tuck,X)).  
the(X,king(X),the(Y,friar(Y),knighted(X,Y))).
```

and the relevant background knowledge is that Henry III is a king, Tuck is a friar, and Robin Hood is neither one.

The output from `re so lvepanapLors` should contain the following formulas (in a list, of course):

```
knighted(henry_iii, robin_hood) .
petitioned(tuck, henry_iii) .
knighted(henry_iii, tuck) .
```

7.5.4 Plurals

The correct semantics for natural—language plurals is still a matter of debate. Webber (1983) points out that in *Three boys bought five roses*, each plural NP has three readings:

- u DISTRIBUTIVE (there were 3 boys and each bought 5 roses);
- COLLECTIVE (the 3 boys, as a group, bought a group of 5 roses);
- CONJUNCTIVE (a total Of 3 boys bought roses, and a total of 5 roses were bought).

The distributive reading is what our quantifier rules already give us, and the conjunctive reading could be derived from it by a transformation not unlike quantifier raising.

The collective reading is the interesting one because it introduces a new concept: SETs or coLrECTIVES. The key idea is that at least on the collective reading of

Three men sang.

and possibly on all three readings, the set of three men is itself a discourse referent, with several attributes:

- ELEMENTs (although in this sentence they are not identified); u CARDINALITY (the number of elements, in this case 3); and

- DISTRIBUTED PROPERTIES, i.e., properties that all the elements share (in this case
 $\forall x)man(x)$).

Figure 7.4 shows a strategy for representing collectives in a knowledge base. Note that collectives can be denoted by conjoined singulars (such as *Curly, Larry, and Moe*) as well as by plurals.

Exercise 7.5.4.1

Give formulas for the distributive and conjunctive readings of *Three boys bought five roses*.

Exercise 7.5.4.2

Consider the knowledge base:

```
collective(x(4)) .
element(x(4), curly) .
element(x(4), larry) .
```

```

collective(x(1)).
element(x(1),curly).
element(x(1),larry).
element(x(1),moe).
cardinality(x(1),3).
sang(x(1)).

```

Three men sang (together).

```

collective(x(2)).
distprop(x(2),X'man(X)).
cardinality(x(2),3).
sang(x(2)).

```

Some cats howled [together].

```

collective(x(3)).
distprop(x(3),X'cat(X)).
howled(x(3)).

```

Figure 7.4 Representation of collectives in a knowledge base.

This implies that Curly is a man, but the query ‘?- man(curly).’ does not succeed from it.

Define a predicate `prove/1` that attempts to satisfy any query, not only by executing it in Prolog in the usual way, but also by making inferences from `distprop` and `element`. Your code should have the form

```

prove(Goal) :- call(Goal).
prove(Goal) :- ...something else...

```

e query ? – prove (man (our ly)).’ should succeed using the knowledge base above.

”.

Exercise 7.5.4.3

The middle knowledge base in Figure 7.4 asserts that the *group* of three men sang. What would it mean if instead of `seng (x (2))` it said `disCprop (x (2), X"sang (X))`? Using this as a hint, how could you represent the distributive reading of *Three million men sang* without using three million discourse markers?

7.5.5 Mass Nouns

If plurals are a puzzle, mass nouns such as *water* or *gold* are an even bigger puzzle. In sentences like

Gold is an element.

This ring is made of gold.

it makes sense to treat *gold* like a proper name: there is only one substance called *gold* in the entire universe, and these sentences make assertions about it (cf. Chierchia 1982). So they can be rendered in a knowledge base as something like this:

```
element(gold) .
```

```
made_of(x(5),gold) .    % x(5) is discourse marker for the ring
```

But in other instances a mass noun denotes a PORTION of a substance (Parsons 1970). A portion is somewhat like a collective except that it has no elements and no cardinality. Instead it has a QUANTITY, which maps onto real numbers using standard units. Such an interpretation is necessary in order to represent sentences such as:

There is an ounce of water in the glass.

```
portion(x(1)) .                                quantity(x(1),29.6,mL) .
distprop(x(1),X'water(X)) .                    in(x(1),x(2)) .
```

% comprising 29.6 milliliters ...
% x(1) is a portion of water % and is in x(2), the glass.

Often, only relations between quantities are known, not actual values:

There is more water in the glass than in the cup.

Finally, note that mass nouns can often be coNVERTED (changed without alteration of form) into count nouns denoting kinds of the original substance: *the wines of California, the heavy metals*.

The semantics of mass nouns is an area of ongoing research. Ojeda (1991) and the papers in Pelletier (1979) describe a number of current approaches.

Exercise 7.5.5.1

Identify the underlined noun in each of the following examples as either a count noun, a mass

Every cat eats meat.

We got some cat food at the store.

The cat food was mainly made of tuna.

Exercise 7.5.5.2

How might you express *John has more money than Jack does* in Prolog, sticking as possible to the formalism used in this section?

7.6 NEGATION

7.6.1 Negative Knowledge

So far we have said nothing about how to represent negative statements such as

Fido does not bark.

We're handicapped by the fact that Prolog itself has no way to encode negative edge. Extensions of Prolog that do so have been developed but are beyond the this book.⁹ One approach is to proceed as follows:

- Store negative facts in the knowledge base explicitly: `neg (barks (fido))`
- Define a procedure that creates the **coMPLEMENT** of each query by adding `neg` is absent, or removing it if it is present.
- Answer each query by trying to prove both the query itself, and its complement. This gives any of four results: *yes*, *no*, *don't know* (neither the query nor its complement succeeds), or *contradiction* (both the query and its complement succeeds).

In a database-querying situation we can get by with something much simpler: **AS FAILURE**, the approach used by Prolog itself. We can assume that a query is false if it cannot be proved true. This is sufficient to answer queries such as:

Is there a dog that does not bark?

`?- dog(X), \+ barks(X) .`

and is the approach that will be used

here. Exercise 7.6.1.1

Assume that you are using an extension of Prolog with explicit negation as described

- (a) What is the complement of `barks (fido)`? Of `neg (barks (fido))`?

(b) Given the knowledge base

⁹8ee Nute (1988); Covington, Nute, and Vellino (1988, ch. 11); Naish (1986); Pearce and

```
barks(fido).  
howls(felix).  
neg(howls(leo)).  
neg(howls(felix)).
```

what answer ('yes,' 'no,' 'contradiction,' or 'don't know') should you get to each of the following queries?

```
°- barks(fido).  
°  howls(fido).  
?- howls(leo).  
?- howls(felix).
```

Exercise 7.6.1.2 (project)

Implement an extension of Prolog with explicit negation.

Exercise 7.6.1.3

Why is contradiction impossible in ordinary Prolog?

7.6.2 Negation as a Quantifier

Consider now the sentence:

No dog barks.

Here no is a quantifier, and we can render this sentence into logic as:

`no (X, dog (X) , barks (X) )`

(‘there is no X which is a dog and barks’).

More formally, `no (Var , Scope , Res )` is true if and only if there is no value of Var that satisfies Scope and Res. In Prolog:

`no (_, Scope, Res) :- \+ !Scope, Res) .`

To get the truth value we do not need to identify Var, nor to distinguish scope from restrictor. We do these things only so that, during the structure-building process, no can be handled like the other quantifiers.

Negation of the main verb (with *nor* or *does not*) works like *no* except that it has the whole sentence within its scope, thus:

Max does not bark.

As expected, no participates in scope ambiguities. An example:

All dogs do not bark.

(1) `all (X, dog (X) , no ( _ , true , barks (X) ) )`
All dogs are non-barkers.

(2) `no ( _ , true , all (X, dog (X) , barks (X) ) )`
Not all dogs bark.

Ambiguities like these sometimes confuse native speakers, and some people are with any quantifier raising that involves negation.

Exercise 7.6.2.1

What does *All that glitters is not gold* normally mean? Could it be interpreted as something else? Explain why it is ambiguous.

Exercise 7.6.2.2

Extend your parser from Exercises 7.4.1.1, 7.4.3.1, and 7.4.3.2 so that it can of the form:

Is it true that no dog barks?

Is it true that every cat chased no dog?

Is it true that no dog chased every cat?

Here you can treat *is it true that* as a prefix that turns any statement into a question.

Exercise 7.6.2.3

Does *Doesn't Fido bark?* mean the same thing as *Is it true that Fido does not bark?* If not, what does it mean, and what is the function of the negative marker (n'f)?

7.6.3 Some Logical Equivalences

Negation gives us many ways to create formulas that are logically equivalent to each other. The most obvious is DOUBLE NEGATIONS

$\text{no}(\_, \text{true}, \text{no}(\_, \text{true}, S)) \quad S$

where S is any formula. There are also interactions of negation with quantifiers:

$\text{some}(V, R, \text{no}(\_, \text{true}, S)) \quad \text{e} \quad \text{no}(\_, \text{true}, \text{all}(V, R, S))$
Some dogs do not bark *Not all dogs bark*

$\text{ali}(V, R, \text{no}(\_, \text{true}, S))$	$\times$	$\text{no}(\_, \text{true}, \text{some}(V, R, S))$
<i>All dogs do not bark</i>		<i>It is not true that some dogs bark</i>
<i>An dogs are non-barkers)</i>		<i>[No dogs bark)</i>

and perhaps most importantly of all,

$\text{no}(\_, \text{true}, \text{some}(V, R, S)) \text{ on } \text{no}(V, R, S)$

Notice that this is not quantifier raising; no ambiguities or changes of meaning are involved. The formulas that we are interconverting have exactly the same truth conditions.

As long as negation occurs only in database queries, these alternative forms are not a practical problem; the inference engine will get the right answers with any of them. But if we want to store negative information in the knowledge base, it is important to convert each formula into a standard form so that the same information will always be expressed the same way.

What to use for a standard form is up to the implementor. One could choose to move all negatives to the outermost, or perhaps the innermost, possible position. Another possibility is to eliminate a quantifier: any system that has no and al l can do without some, or if it has no and some it can do without at 1.

Exercise 7.6.3.1

Simplify the formula

$\text{no}(\_, \text{true}, \text{some}(X, \text{dog}(X), \text{no}(\_, \text{true}, \text{bark}(X))))$

to the simplest logically equivalent form. (To avoid bumping into a discrepancy between our definition of al 1 and the standard one, assume that there is at least one dog in the knowledge base.)

Exercise 7.6.3.2

Define a predicate `simplify / 2` that will do the previous exercise for you. That is, define a predicate that will “simplify” a formula containing negation, as follows:

- Transform double negation, `no (_, Ctrue, no (_, Ctrue, S))`, into `S`.
- If no occurs in the scope of another quantifier, move it out and change the quantifier so that:
`some (V, R, no (_, true, S))` becomes `no (_, true, all (V, R, S))` and
`all (V, R, no (_, true, S))` becomes `no (_, true, some (V, R, S))`;
- Do both of these things recursively; that is, before simplifying any formula, attempt to simplify its scope.

Exercise 7.6.3.3

Of the many available introductions to first—order logic, that of Barwise and (1991) meshes especially well with the material covered in this chapter. It covers such as Skolemization that are left out of more traditional texts. MCCawley (19s1) also useful because of its length (Its explanations are fuller than usual) and because its emphasis on natural language semantics.

Aside from logic texts, books on semantics are of two kinds: some cover word meanings (Palmer 1981) while others treat syntactic and lOgical issues. A introduction of the latter type is Chierchia and McConnell—Ginet (1990).

A good example of semantic analysis in action is Horn (1989), a but readable study of negation that treats many other phenomena along the way. (1979) gives insightful analyses of a wide range of phenomena, many of which can be adapted into frameworks other than Dowty's.

For an introduction to model theory, see Bach (1989). The classic paper on eralized quantifiers is Barwise and Cooper (1981), but Peres' account (1991) is and more accessible to the beginner.

7. Conclusion and Future Work

Introduction

In this article we offer a starting point for dialogue on the opportunities offered by social robots and the potential impact on your organization. We start by identifying what a social robot is and review some key robot statistics. We present key drivers for the adoption of social robots and the economic sectors where we expect to see their introduction over the coming years. We want to build awareness on the increasingly complex social roles that robots may be able to fulfil in the future and we offer a glimpse inside the minds of four Founder / CEOs that are building robot companies.

Contents

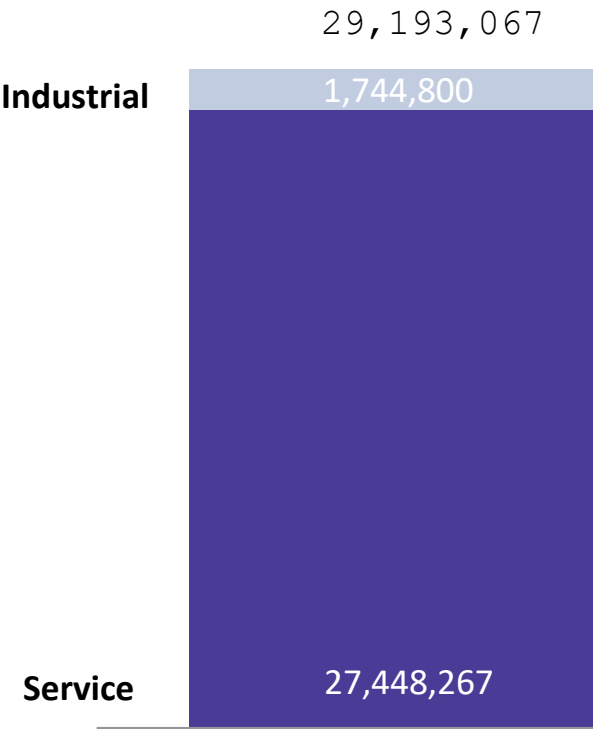
- A. > A new type of robot is coming to town
- B. > Robot categories and key statistics
- C. > Social robot adoption: key drivers and leading sectors
- D. > As their dexterity improves, robots take on increasingly complex social roles
- E. > Do you want to explore the potential of social robots in your organization?
- F. > Get to know 2016's new robot breeds and their makers
- G. > Would you like to learn more?
- H. > Robot roll-call

I. > Sources

Robots are programmable devices that automatically perform complicated and often repetitive tasks. If you heard about a robot that could interact with you and become a helpful aid in your daily life, would you want to learn more? If so, 2016 is going to be a year to remember, because the first batches of social robots are shipping right now.

What is a social robot? Robots are already a familiar sight in our factories and warehouses, where they continue to win terrain. These robots, however, typically operate in highly structured environments and have only limited interactions with humans. Indeed, their arrival has often led to the replacement of the human workforce altogether. Now, with an accumulation of innovations in artificial intelligence (AI), sensors, and battery technology, robots have acquired the ability to enter our domains at work, home, and in the public sphere. Social robots are autonomous mobile machines that are designed to interact with us and exhibit social behaviors such as recognizing, following and assisting their owners and engaging in conversation.

Number of active robots



2

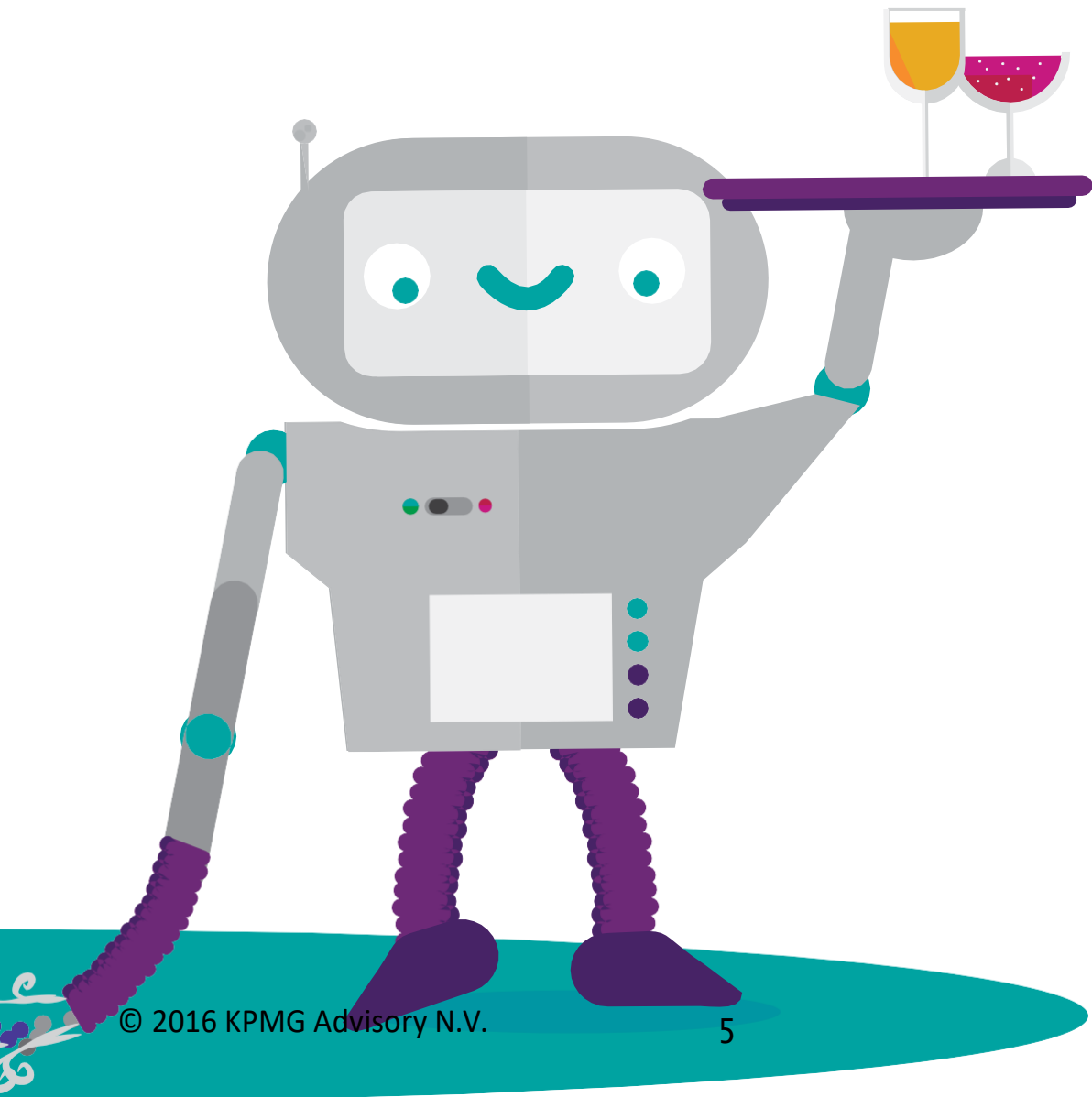
Earth

Mars



The movie 'I, Robot' has an interesting perspective: fast forward to the year 2035. The streets are filled with humanoid robots that assist people with their day to day activities, while humans in need have been given robotic limbs. Yet, before we explore the future, it may be helpful to review some of today's statistics. The International Federation of Robotics classifies robots into two

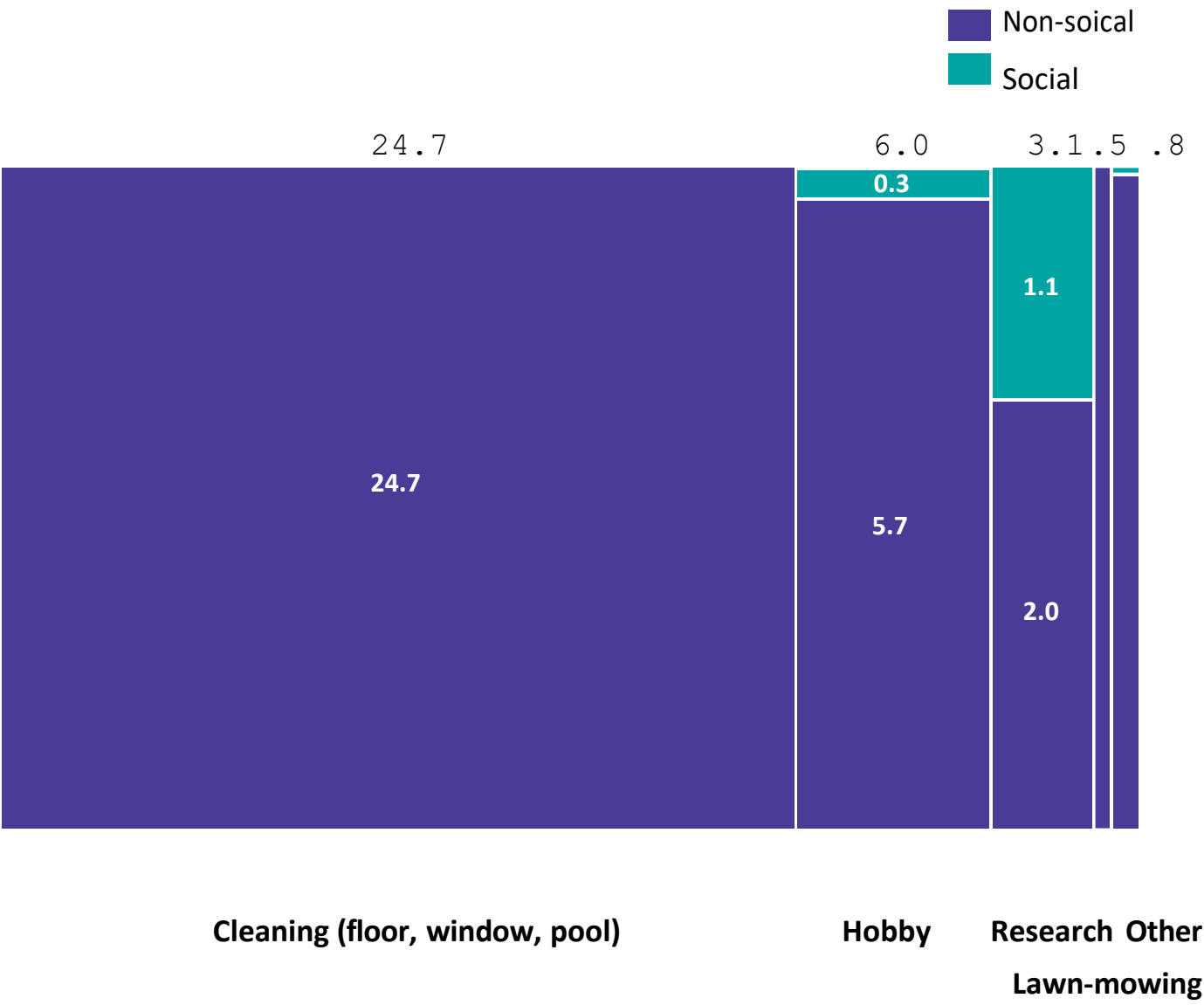
categories: industrial (e.g. car manufacturing) and service. The latter is further divided into professional (e.g. milking robots) and personal (e.g. vacuum cleaners). Currently, there are about 1.7 million operational industrial robots globally¹. In total, there are ~30 million active robots, the majority of which are vacuum cleaners. This means there are ~250 humans to every robot. The industrial stock is set to grow by 35% over the next two years. In addition, based on company forecasts, more than 150,000 professional service robots have already been, and will be, installed in the period 2015-2018. Sales of personal service robots are estimated to be ~35 million units between 2015 and 2018, with ~1.5 million of them having social robot characteristics. More than 7,000 Pepper robots, for example, have been bought for homes across Japan since they went on sale there in June 2015².



© 2016 KPMG Advisory N.V.

5

Personal robot sales 2015 - 2018 (million units)

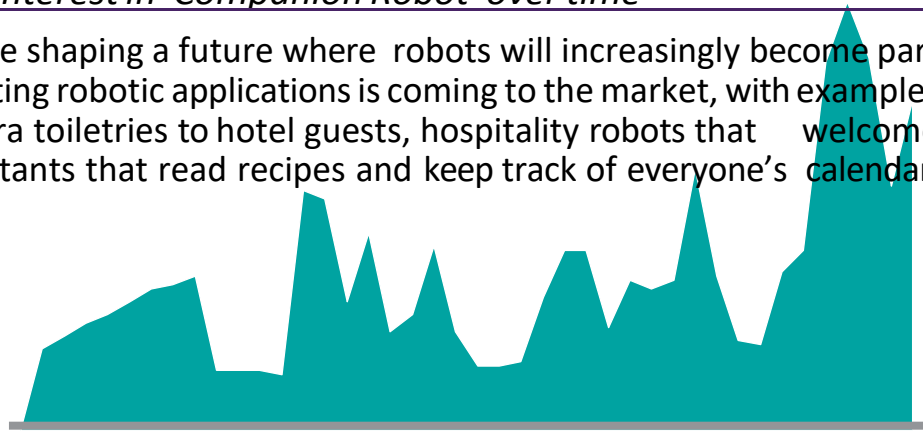


Forecasting the diffusion of new technology is notoriously difficult. A compounding factor when it comes to robots is the complexity of their systems, which typically include refined mobile hardware and AI software. Nevertheless, there are a number of indicators signaling that the time for rapid robotic progress is upon us. First, breakthroughs in AI are being made frequently. Self-driving cars are a much cited example, and think also of how Google's Deep Mind has recently learned how to beat one of the best human 'Go' players; Go is more complex than chess and the expectation was that man would continue to reign supreme for the coming decade. Second, continued miniaturization efforts and progress in material sciences has led to the most intricate mechatronics yet such as robot hands that move in the same way as ours.

Third, the price of robots continues to fall, and the new breeds presented in this article have price points that make them accessible to many different constituencies. Although Moore's law arguably no longer applies, the cost of key robot components like sensors is still falling. The average price of sensors has halved over the past decade, all while their quality has improved (see graph: Average sensor cost). Public venture capital investment in robotics also doubled in 2015, signaling a tipping point that is set to drive continued development (see graph: Public venture capital robotics investments). Finally, there is anecdotal evidence that society is more open to accepting robot companions, not just in Japan³, but globally (see graph: Interest for 'Companion Robot' over time).

Interest in 'Companion Robot' over time

Dozens of innovative companies are shaping a future where robots will increasingly become part of everyday life. A wave of interesting robotic applications is coming to the market, with examples such as bell-hops that deliver extra toiletries to hotel guests, hospitality robots that welcome customers to a store, family assistants that read recipes and keep track of everyone's calendar, and robotized walkers that



Oct- Dec- Feb- Apr- Jun- Aug- Oct- Dec- Feb- Apr- Jun- Aug- Oct- Dec- Feb- Apr- Jun- Aug- Oct- Dec- Feb-

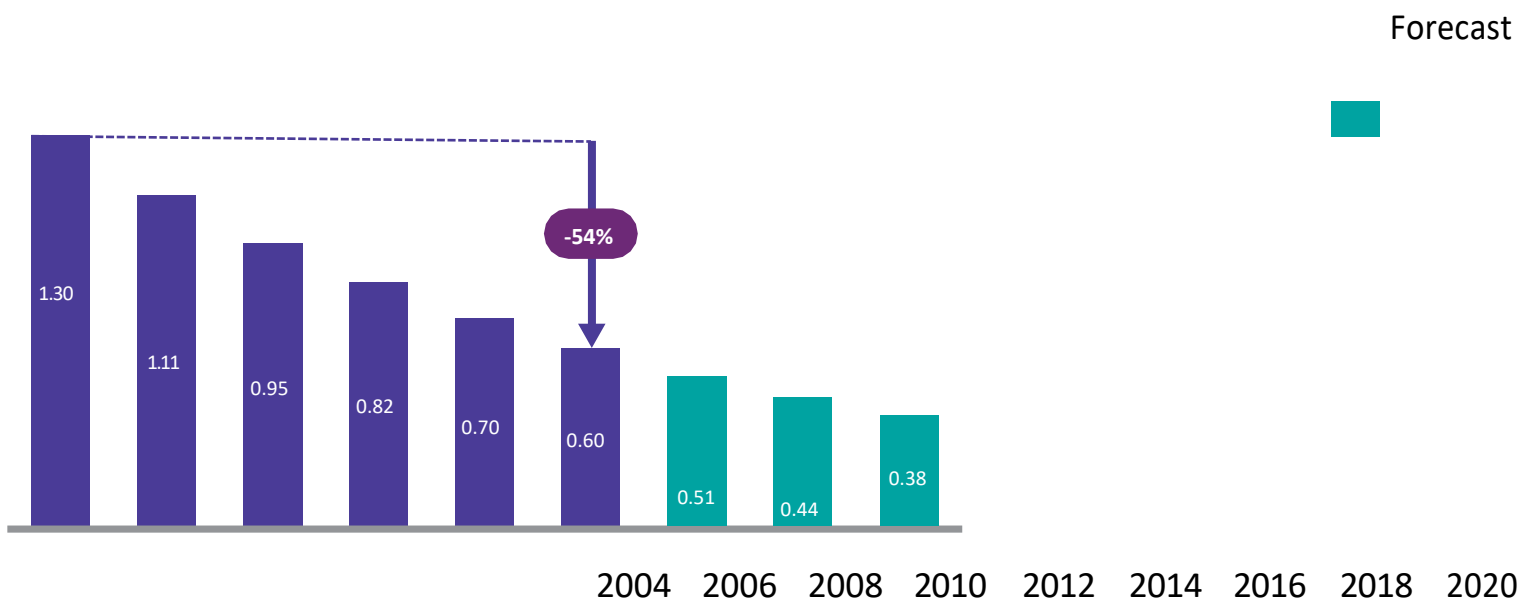
enable the elderly to adopt a healthier lifestyle. This article contains a robotic rundown which showcases several of these robots and their makers. By definition, this list is not

12 12 13 13 13 13 13 13 14 14 14 14 14 14 15 15 15 15 15 15 16
complete: there are more

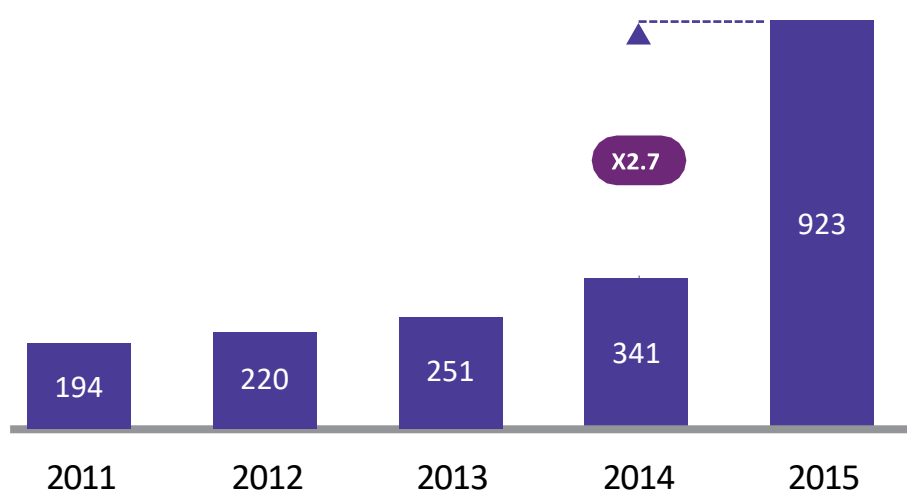
Average sensor cost (\$)

social robots in development and software updates can potentially and retroactively make existing robots more social going forward.

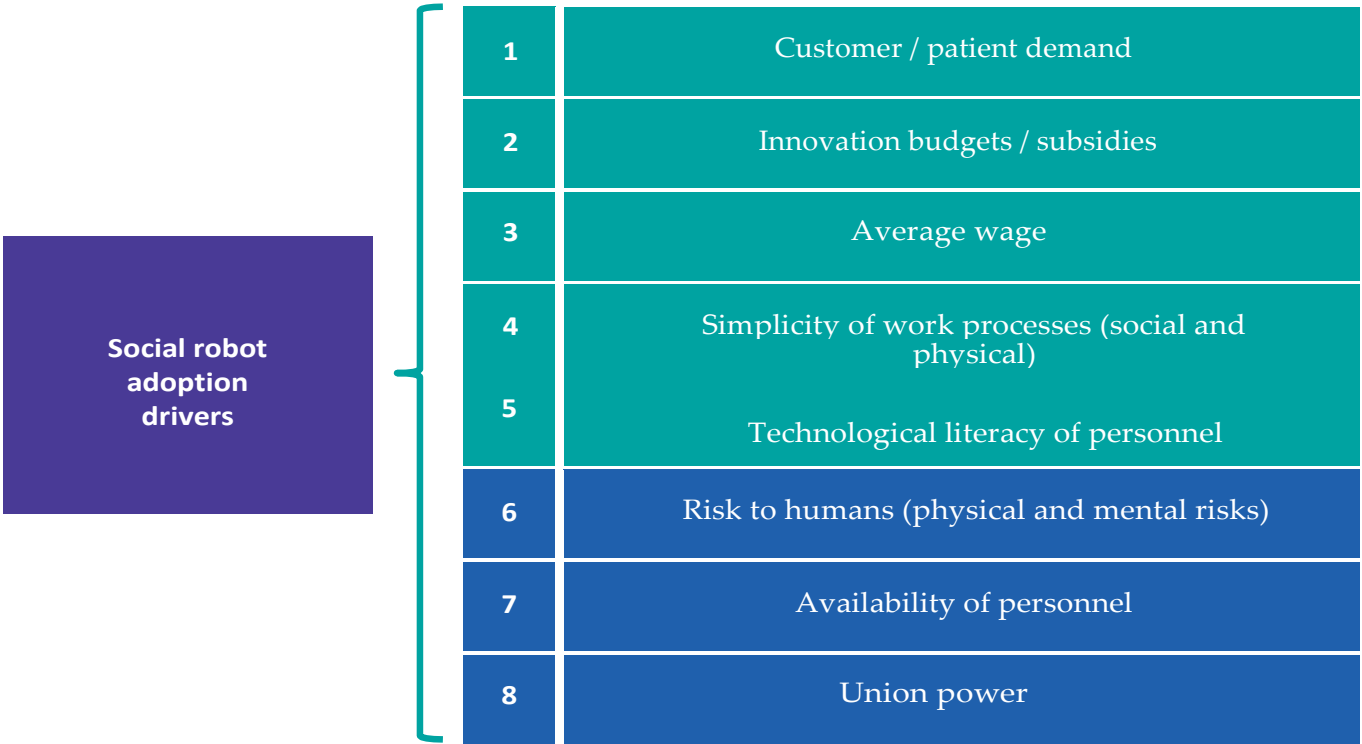
We believe that the robots considered here are illustrative of applications that will emerge over the coming years. Our goal with this article is not to make exact predictions or steer readers towards deploying these new technologies. Instead, we hope that this article will act as a conversation starter so that leaders in this field can explore the potential impact on customers, colleagues, organizations, and society as a whole.



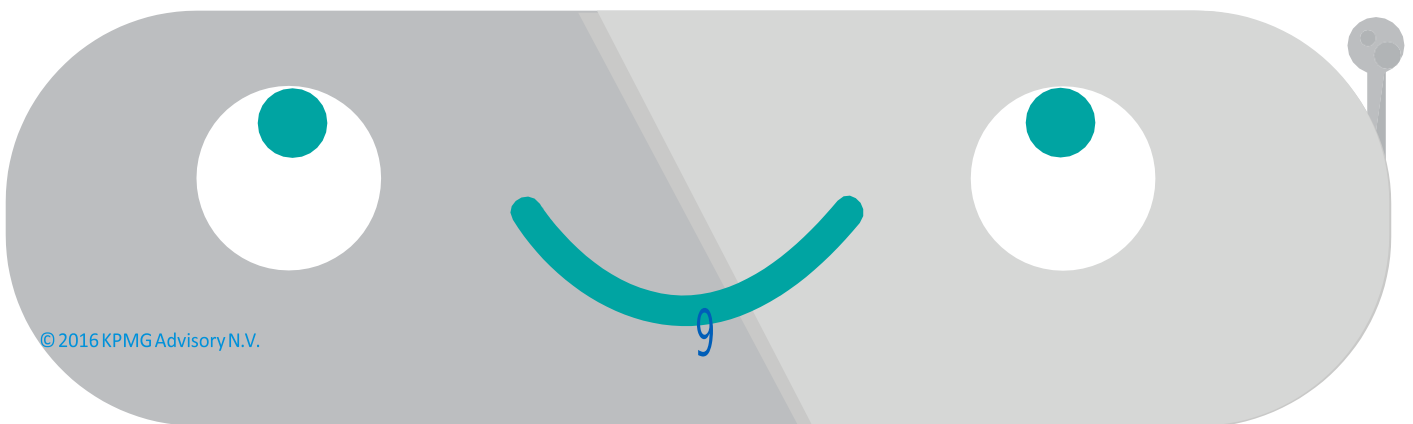
Public venture capital robotics investments (\$ million)



We have identified eight key drivers for the adoption of social robots. These can be used to assess to what extent your industry, organization or even specific departments/teams can be a fertile breeding ground for social robots.



- 1.** Organizations whose customers or patients demand robotic solutions are expected to lead the way in the adoption of social robots. Think, for example, of people asking for the adoption of robotic walking assistants that also remind their elderly parents when to take their medicine (see LEA in the interview section). Initial reports from the deployment of Relay (a robot built by Savioke) indicate that hotel guests are so happy to see it that they instantly share pictures of it on social media.
- 2.** Another driver is the availability of innovation budgets or subsidies. Organizations that allocate funds for innovation are more likely to experiment with social robots and explore the implications for their processes and potential business model or value chain position. Alternatively, subsidies can drive the adoption of social robots by both established players and start-ups. Over the last few years, for example, research projects in robotics for the elderly have had a total budget of €50 million funded by the European Union⁴.
- 3.** High average wages can encourage business leaders to investigate options for (partly) automating certain work. This then releases some of the capacity of the human



workforce to focus on tasks where we have an advantage (such as creative thinking). Social robots, like other types of automation, can be both complementary to or a substitute for conventional labor. In an ideal world, automation would result in greater productivity, thus potentially leading to higher wages. Graetz and Michaels (2015)⁵ have found that increased robotic automation leads to higher added value and labor productivity, although there is also some evidence of the crowding out of low and middle-skilled workers (in factory work). The future effects on productivity and labor aside, a broader value case conversation is required when deploying social robots.

4. The simplicity of work processes can drive the adoption of social robots. When the process at hand is relatively simple to execute, it can be automated more easily. Conversely, if the logic required to execute the process is complex, it may be harder to automate. Tasks which occur in a structured environment are easier to automate than those in an unstructured setting. Interactions with humans that require a complex set of social rules are more challenging when it comes to automation than those that are merely transactional

(compare, for example, the delivery of an additional towel to a negotiation to help someone take their medicine).

5. Another key factor for the adoption of social robots is the technological literacy of their future colleagues. When workers have positive experiences with using technology like computers, tablets, and software to make their lives easier, they will be more inclined to allow social robots into their midst as they have a positive perception of the usability and ease of use of the new technology.

6. The real and perceived risks social robot deployment poses to humans can drive down their adoption rate. In terms of real risk, it is helpful to distinguish physical and mental risks. The former is the potential damage caused by a robot interacting with a person in an unintentional manner. There have been several dramatic incidents with industrial robots, for example⁶, with a Korean woman 'attacked' by her vacuum cleaner in 2015⁷. Mental risks are more subtle. Compare, for instance, the apprehension some elderly humans feel about using their hearing aids. Studies indicate that a factor in this resistance is the fear of being regarded as incompetent⁸.

Robots designed for the elderly may thus suffer from a similar stigma, with elderly patients seeing the use of such a robot as an indicator of the loss of some of their previous ability. Alongside these real physical and mental risks is the perceived risk of social robots. Reports from the first deployments of the social robots discussed in this article are generally positive, i.e. people enjoy interacting with these machines. However, the public discourse is increasingly centered on what has been described as the 'A.I. panic'⁹. This issue revolves around machine intelligence with a 'recursive self-improvement' capability. Business and academic leaders have spoken of the danger of sentient AI, which may eventually result in the extinction of the human race. This negative discourse may instill a fear of robots in some individuals. It should be noted that those at the forefront of developing

AI code generally have a much more nuanced view. To quote Andrew Ng, Chief Scientist at Baidu: “There’s a big difference between intelligence and sentience. There could be a race of killer robots in the distant future, but I don’t worry about AI that turns evil today for the same reason I don’t worry about the problem of overpopulation on the planet Mars.”¹⁰

7. Availability of personnel: if there is a high availability of workers, there will be a lower drive to deploy social robots. We note there can be a causal relationship between wages and availability of personnel (e.g. paying higher wages leading to more people willing to do the job). However, it can be difficult to find people to undertake certain tasks, such as those that require expert skills or occur at unfriendly hours or in more dangerous /unwelcoming environments.

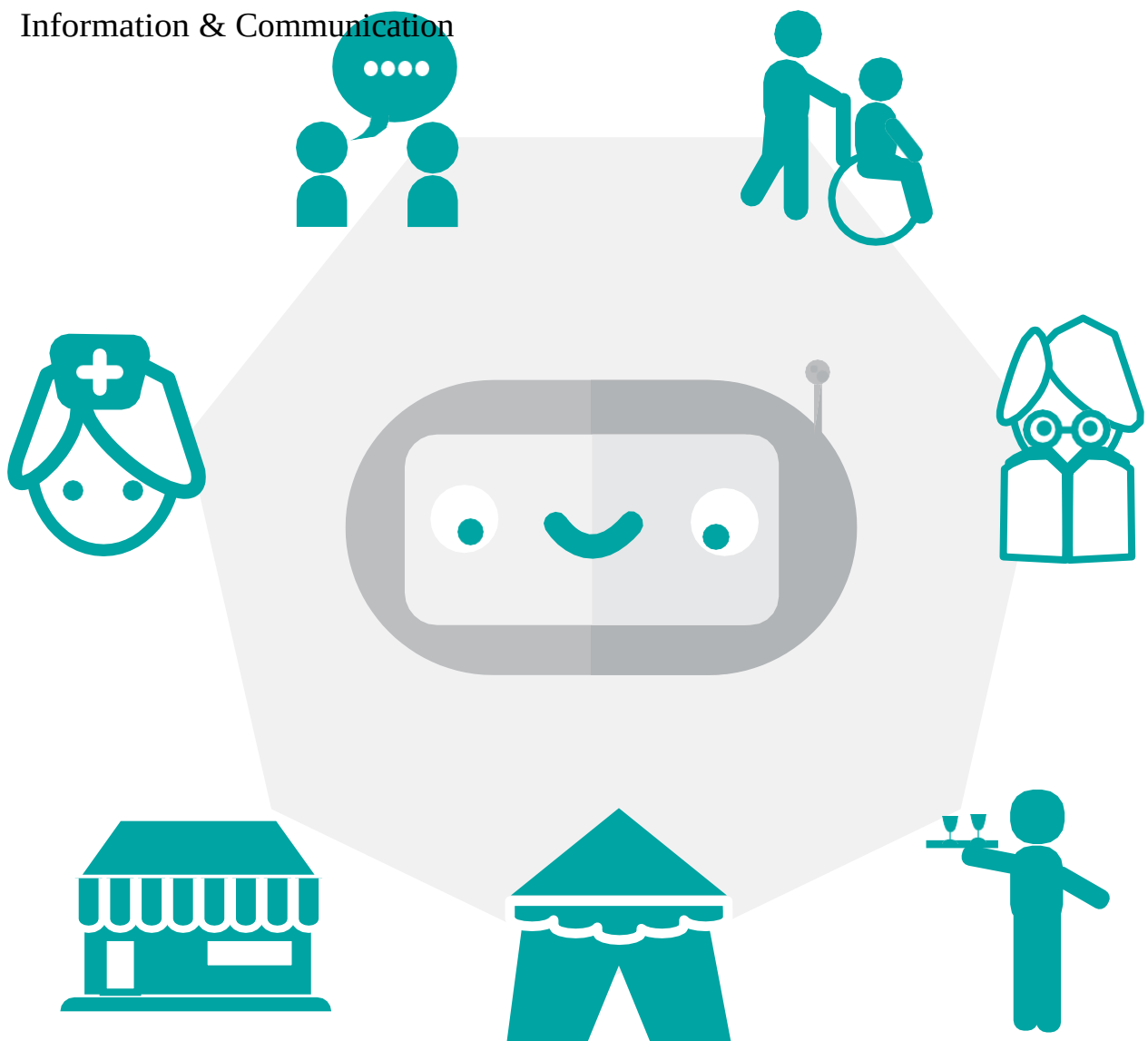
In those situations that lead to a low availability of personnel, using robots may be an attractive option.

8. Finally, another factor that may limit the adoption of social robots is the power held by trade unions. While some unions are modernizing and more open to considering the potential benefits of automation, unions in general tend to oppose any technology that may replace (parts of) jobs¹¹. Based on the drivers above, we have reviewed twenty-one NACE codes (framework used by the European Community to collect statistical data on economic activities) and identified seven sectors where we expect innovative organizations can lead the way in the adoption of social robots. It can be helpful to take a tailored snapshot for your organization.

“Idon’tworkonnotturningAIeviltodayforthe samereasonIdon’tworryabouttheproblemof overpopulationon Mars.”

7 sectors to lead the way in social robot adoption

Information & Communication



**Health &
social work**

Healthcare

Healthcare

Education

SHOP

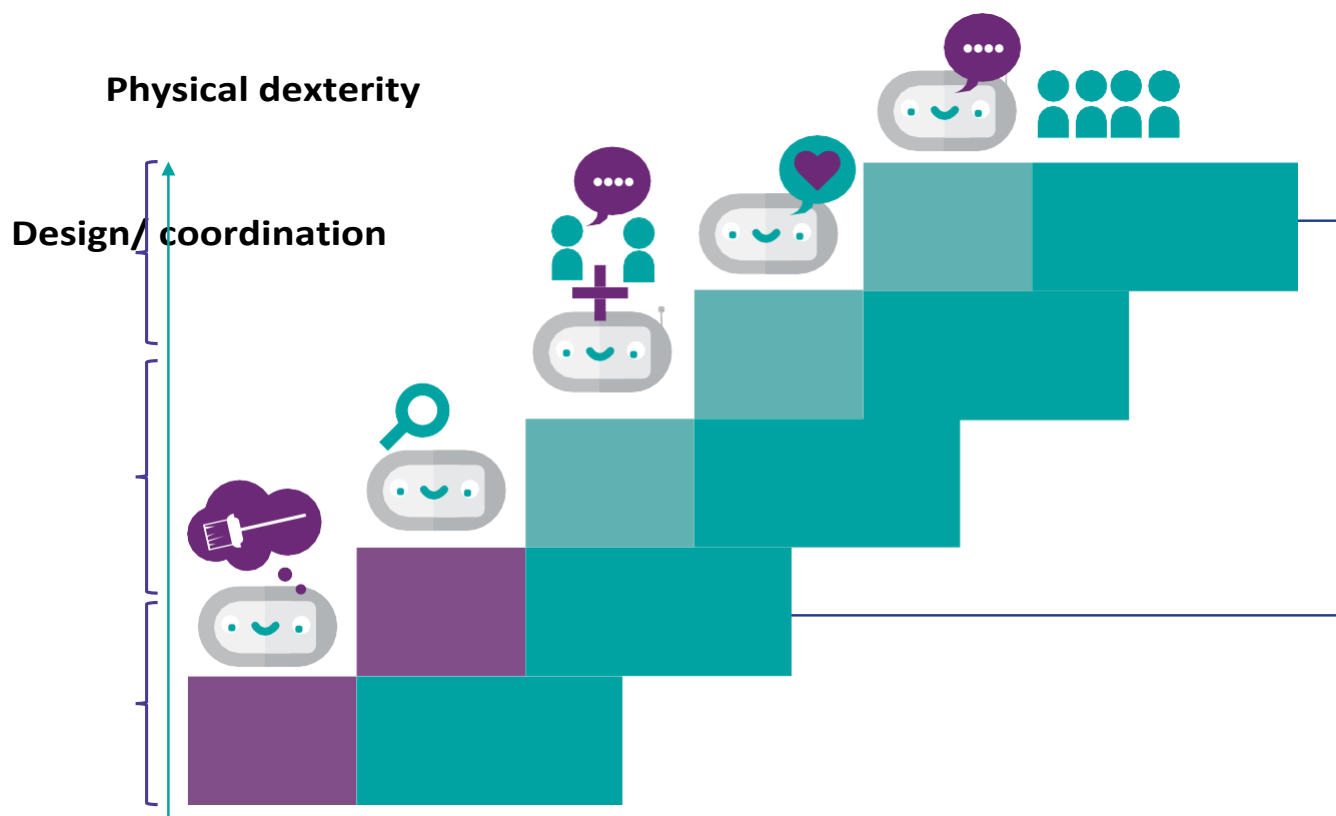
Retail trade

**Arts &
entertainment**

Accommodation & foodservice

With advances in AI and physical robot designs, these machines are taking on an increasingly complex array of social roles. We have produced the diagram below to initiate dialogue on the changing types of role that can potentially be fulfilled by robots in our homes, workplaces, and public spaces.

Social robot roles in human society



Coach

Learns with you and shows a better way (ability to improvise)

2050?

Friend

?

Understands your feelings and provides advice

Manipulation/ construction

Teammate

Group problem solves in structured situations

2016

Perception/

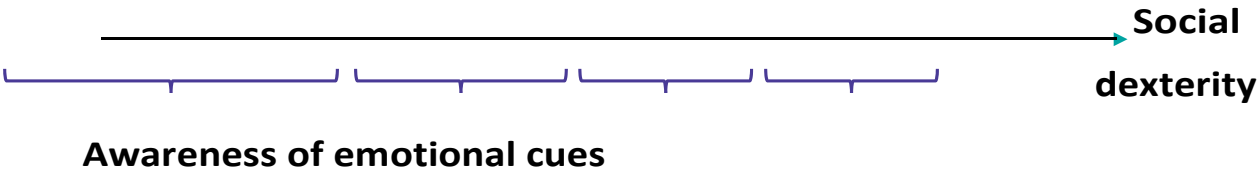
Helper

Senses emotional state and tailors
interaction style

navigation

Concierge

Talks to you and
points you in the right direction



Negotiation/ persuasion

At the start of the social dexterity (i.e. handiness) axis is the absence of any social relationship. Instead, the interaction is purely transactional and the robot does not take up any social space (think, for example, of the I.V. station¹⁴, which is a robotic station that mixes intravenous solutions for use in hospitals). As the social complexity of a task increases, robots may need to be aware of emotional cues such as joy, excitement, or anger. Think, for instance, of a robot sales assistant that reacts to the delight of customers at the demonstration of a certain product by explaining additional characteristics or signaling interest to a human nearby to 'close the sale'. Even more complex social settings would require robots that are able to negotiate with human actors: imagine a robot that finds creative ways of encouraging the elderly to leave the house or start an exercise routine. The next wave in social dexterity may include robots that can build on their emotional awareness and negotiation skills to develop affective relationships with the user. In the ultimate socially complex setting, a robot would need to have the ability to learn by applying existing knowledge in a new environment and then transfer that learning to its human counterpart. It should be noted that, currently, AI is only able to 'see' basic emotional cues when they are expressed

very clearly, meaning that robots are not yet able to pick up on the many nuances of typical human interactions.

Physical dexterity relates to the difficulty of what is required to happen in the physical environment. At the start of the physical dexterity axis are devices that cannot ‘see’ the world and are immobile (e.g. Amazon Echo). At the most basic levels are tasks that require perception and navigation, such as delivering a toothbrush to a hotel guest or playing hide- and-seek. More complex tasks involve the manipulation of objects or construction work, such as those found in cooking processes or clearing trays from a table. The most complex physical tasks require the design of new objects, the capacity to coordinate multiple objects, and the ability to develop new solutions through improvisation/developing logic.

Note that our diagram shows the progression of machines that are increasingly dexterous, both physically and socially. In reality, the interfaces we choose for social AI purposes may be very different. Think of an evolved version of AI assistants like Siri/Cortana/ Alexa that live in the cloud and interact with you via your mobile phone/television set/ home computer(s). One can also imagine devices that have no physical dexterity, but



are very social (a 2025 version of Amazon Echo perhaps?¹⁵), while, alternatively, we may choose to develop machines that completely mimic, or even improve upon the human physical form, but are socially as limited as theme park animatronics.

Also note that some robots can already ‘act out’ some aspects of the roles depicted. A humanoid robot that demonstrates a morning physical routine for the elderly or portrays yoga poses acts as a coach, for example. The affective relationships that some owners have developed with their AIBO robot dogs are also well documented^{16,17}.

So, when will human minds become smart enough to develop synthetic minds that match our own? This type of intelligence is referred to as High Level Machine Intelligence (HLMI), and there is some consensus on when you will have the opportunity to mingle with robots that possess it. Vincent C. Müller and

Nick Bostrom conducted a survey in 2013¹⁸ that allowed 170 experts to predict when they expect HLMI to arrive. The median estimate of the respondents was that there is a 50% chance that HLMI will have been developed around 2040-2050, rising to 90% by 2075.

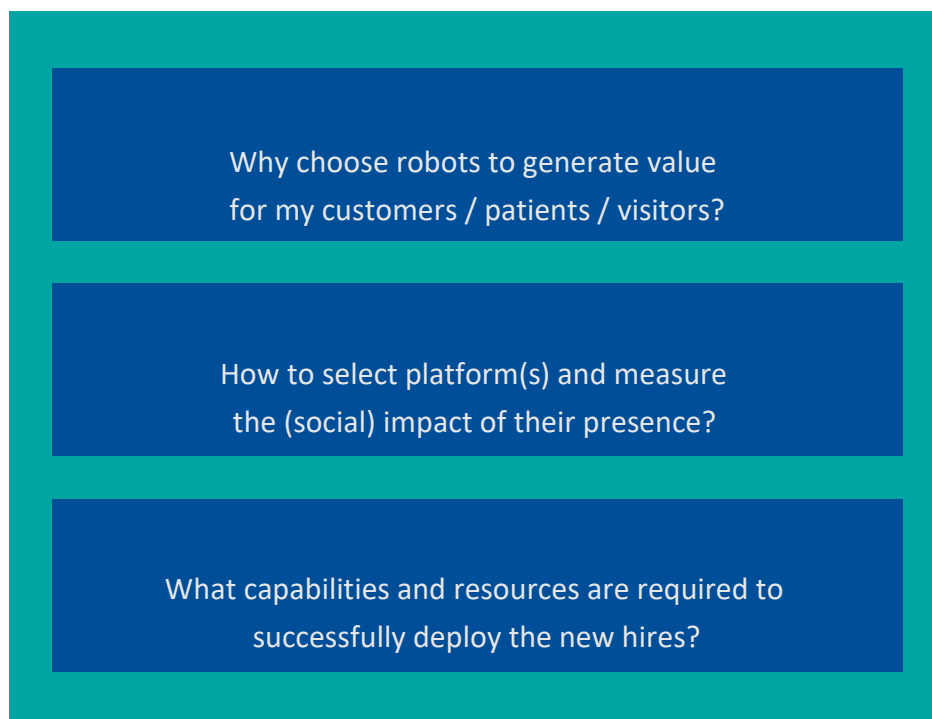
In light of this prediction, it is important to highlight three perspectives that are shared by the four robot company founders/CEOs we interviewed for this article:

- 1. Robots will become part of our lives long before we develop High Level Machine Intelligence – current Artificial Intelligence is limited
- 2. Be careful when designing the social aspects of robots: humans need to remain in charge of social relationships
- 3. Focus on developing robots with use-cases that work and add value today

Want to explore the potential of social robots in your organization?

Social robots can potentially deliver great value and therefore hold great promise for business and society. Through learning effects, early adopters stand to gain competitive benefits in terms of quality, efficacy, durability, speed, and cost. Moreover, deploying social robots may unlock significant value gained through human-robot interactions. Robots and humans can learn to work together and complement each other. Think, for example, of a Da Vinci surgical robot, but now with strong AI suggesting alternative options to the surgeon. Or, in the more commercial arena, a robot that senses why a customer is hesitant and suggests alternative approaches to the human salesperson. Further, in the world of elderly care, a robot can encourage its owner to keep moving and visit friends, creating a virtuous cycle of healthy behavior.

When you want to explore how social robots can help there are three questions to answer:



We argue that business leaders require a compelling story to convince organizations to explore the use of social robots (including the value case). To optimize the social robot added value, we propose piloting robot platforms and experimenting with new processes in a structured manner, measuring a relevant set of KPIs. Finally, changing (service) processes can be challenging and time consuming, for example, workers may require training and other tools if they are to work effectively with social robots.

Robot Care Systems



Robot Care Systems builds robots for elderly care purposes. The company was founded in 2014 and is headquartered in The Hague, the Netherlands. Its mission is “To provide a little help to keep your independence as you get older. You can continue to enjoy life.” In 2015, the company won the Shell Livewire and the Herman Wijffels innovation awards. It has developed the LEA robot (Lean Elderly Assistant), which will be launched this year. LEA provides help with walking and assists users to adhere to their daily routine and, for example, indicates when medications should be taken or that it is time to exercise. LEA thus helps people to lead an active life, and ensures safety and easy communication. The device is packed with sensors to guarantee the safety of its users.



Interview with Maja Rudinac, CEO Robot Care Systems

Why did you and your team build LEA?

“We want to help the elderly to improve their quality of life during their final 20 years by allowing them to remain mobile.”

What can you tell us about LEA’s value proposition?

“It makes people more mobile. It assists people in getting and standing up and helps with the simple things so that care workers can focus on social contact. In short, it makes the elderly more independent, more mobile and safer.”

How are you going to pilot LEA?

“We’ll pilot it in five countries with five month trials that involve all relevant stakeholders: the elderly, municipalities, insurers, physical therapists, nursing homes etc. We’ll test it in two settings: with the elderly who we hope to enable to live alone for longer, and in care homes.”

How will staff know the robot is performing as it should?

“There are a number of indicators on the robot. We’ve also developed an extensive man-

ual and will provide training.”

What are the key lessons from early deployments?

“There’s a major challenge in making a robot that’s robust enough for use in the real world. Performing a task once in the lab isn’t enough; it has to be able to deal with all kinds of practical challenge. Also, the technology needs to be adjusted to people, not the other way around. We’ve concluded that, with robots, simplicity is the ultimate sophistication.”

What are your key market segments?

“We’re focused on elderly care and rehabilitation. In the future, we may develop additional solutions for hospitals (e.g. moving goods) and we hope to make our robots fit for outside use as well.”

How do you see the future of social robotics; how social should robots become?

“This is an important political choice we have to make. I believe robots are primarily a tool and they should become as social as would be required for them to navigate our world. But we, people, should take the lead in social interactions. Robots shouldn’t replace humans in a social setting, but they should augment and complement us. We have to be careful. Think, for example, about chat machines that mimic humans. What if, for some people, there’s no longer a reason to leave the house? Machines shouldn’t change the social contact we have.

There are specific situations where a certain level of attachment to the robot may be desirable, for example in cases of dementia, autism, and for those with mental disabilities.

But let's make sure we're very clear on the following question when we create a new robot: how will it help people or the planet?"

What's your perspective on artificial intelligence?

"The biggest challenge with robots is that they can't apply existing knowledge in a new environment. I don't agree with the current trend in robot design where it seems the objective is to mimic a human. My suggestion is to start small with robots that solve specific problems and then build on that.

Many scientists in the field expect robots to be at a general level of intelligence around 2050. That's when they should be able to compete with the best human players in soccer. But they'll be entering our society much sooner."

What are the key drivers for social robot adoption in your view?

"Robots have the ability to take over repetitive tasks, secure buildings, enter dangerous areas where humans can't or don't want to go. Also, the susceptibility of humans to new technology is key. In general, I find people are quite open to new technology; think about cell phone adoption for example."

How should organizations prepare for the introduction of social robots?

"Be open to new technology. Find ways for humans and robots to work together, with the robot as a helpful tool. It's also key to prepare humans through an appropriate training program."

"technology need to be adjusted to people, not the other way around"



Savioke (pronounced savy oak) wants to enable hotel personnel to provide better customer service. The company was founded in 2013 and is headquartered in Santa Clara in the United States. It is passionate about delivering easy-to-use, yet sophisticated, robots that can help people. The aim is to improve lives by developing and deploying robotic technology in the places where people live and work. Savioke also believes in the power of open source software. The company leverages and contributes to the open source Robot Operating System (ROS) that it helped to develop at Willow Garage. Savioke was given a GameChanger award by the Robotics Business Review. The company has developed Relay, a robot that relays products from one person to another. The first application is in the hospitality industry. Relay is 91.4 cm tall, weighs less than 45kg, and

has a carrying capacity of 0.06m³

Interview with Steve
Cousins, CEO Savioke

You have extensive experience with robotics, why did you choose to develop a robot for the service industry?

“We knew we wanted to continue down the autonomous path and we looked at the cost factor. One of the first questions we asked was: should we be in commercial environments or the home? We landed on the goal to develop an affordable robot for the service industry after we had an ‘aha’ moment. Robots are most helpful when there are repeatable activities that have to occur often. Think, for example, of hotels, restaurants or elderly care. These work environments feature tasks that occur many times throughout the day.

We looked at a number of industries and had to pick one. The fundamental factor we bring is safe and reliable mobility in human spaces. Not getting lost, moving gracefully, and using the elevator. We started in a domain that has three year olds running around. We see an opportunity to leverage the lessons we are



learning in other environments that require a high safety standard.”

What’s your perspective on social robots?

“We set out to build something useful. Robots are there to do work. We recognize that the technology that we build will be treated by people as if it’s a social actor. One question is whether you start to design it to be social. We could also build a robot that isn’t very social. We’re providing a platform for safely and reliably navigating through human spaces. We’ve had robots working in hotels for over 1.5

years now. As we deploy the robot there are all kinds of interaction. The robot has necessary interactions for it to do its work (deliver products) and then incidental interactions (as it encounters people in the hallways).

It all depends on how we can improve guest satisfaction and the goals of the hotel. We started with service robotics and ended up adding the social element. The question came up: should we put eyes on it? We went for the middle ground. We're currently adding a 'mingling' activity to the robot. Our mission is to deliver – be as cute as you can, but do useful work first.

Take speech, for example. We find some people prefer interacting with machines through a touch screen as it creates less social anxiety. Some may expect a speech capability and would want to talk to the machine. Over time, we plan to meet the wishes of people at that level. It's not one or the other. It makes sense for us to add features when we feel the time is right.

The most successful social robots have been the ones where the people they are interacting with have a need for a social agent. Humanoid robots, but not too humanoid, for children with autism, for instance. For example, Paro has been used in Denmark to help dementia sufferers speak again."

What type of metrics do you gather in your current deployments?

"We measure how often the robot is used. Also, the number of deliveries per day and the success rate. We're consistently driving that upward and we're already in the 98% percentile."

What's your view on AI?

"I'm a skeptic about AI. Robots don't have common sense. We try to have a robot that works as much of the time as possible and otherwise it falls back to a human brain. Think, for example, of an issue that we encountered. The robot has to take a delivery to the 7th floor and somebody on the 3rd floor pushes the robot off the elevator. Someone will then remotely restart the delivery and the customer won't even know something was wrong. Eventually we'll solve this autonomously."

How do staff members interact with the robot?

"Staff members can pay attention [to it], but they don't really have to. We position Relay as a service robot; they don't have to worry about it. We give staff the ability to monitor the robot. They can monitor it from their station. But it can also function as a fully autonomous tool. The relay is from one person to another, a baton from a staff member to a guest."

What was the biggest surprise you had based on your deployments so far?

"We were really surprised by guests' delight. Many people go on social media and post selfies taken with the robot. Also, staff reactions were a very positive surprise. Unfortunately, the media has created the idea that robots are here to take our jobs. Actually, when one of the robots was offline, the hotel staff was sad that their 'co-worker was out of service'; they celebrated when he came back online. We're automating tasks, not jobs. People have so much more inherent capability that's released by automating low-value tasks. Societally, ev-

"Hotel staff was sad that their co-worker was out of service and they celebrated when he came back online"

every time we've had more automation, we've had more wealth as a result."

How are you approaching the market?

"We're talking to the brands, talking to hotel owners and managers. A big benefit is that robots in hotels are easy for reporters to see and are highly photographable; we've received an enormous amount of press."

What's your value proposition?

"We provide robots as a service. We perform the installation and integration with the building infrastructure and phone system. We can't drop-ship it at the moment; there's an installation step. In terms of cost, we were able to create a design that comes in at a cost that's significantly lower than the lowest-cost worker. Also, the robot is always available.

Note that because we provide the robot as a service, it's our responsibility to keep the fleet operating. If there are issues, we pay for service visits. So that aligns incentives so as to create a highly reliable, quality product."

Who are your key partners?

"We're looking at the supply chain. We're exploring relationships with contract manufacturers to take on more of the assembly tasks. Then there are installation partnerships with companies that have personnel globally, and we're also exploring sales/distribution partnerships at the moment."

What are the key drivers for Relay's adoption?

"Fundamentally, guest satisfaction drives adoption the most. Service robots cost effectively improve the level of service that a hotel

can provide. They're fast and remove social awkwardness from the equation (how much to tip, having to get dressed). The same approaches will work in other domains as well. For example, hospitals are judged by patient experiences and service levels now."

How should organizations prepare for the introduction of social robots?

"Organizations that succeed are those that manage to improve their service processes. You've never read a resume that says: I can ride the elevator all by myself; I can walk the hall all by myself. Our robot does a couple of things very well, which allows staff to expand what they are doing in terms of service processes."

What's your ambition? What does the impact look like?

"We want to put robots in the world to help people. Helping people do their jobs and provide higher levels of service. Helping people to help themselves. Think about people with disabilities who are completely dependent on caregivers. I once worked with a mute paraplegic who used a robot as his external body. He controlled it by moving his head. He turned the robot towards himself and put the robot arm to his face. We were worried, because we didn't know what he was doing... then he scratched his nose. It was the first time in ten years he was able to do that! This is what robots can do - help people who can't help themselves.

Savioke as a company wants to improve users' quality of life. At the same time, we're a commercial company and are looking for lucrative markets."

The company's goal is to "design and develop robots that are accessible to everyone and help people live easier and safer lives, and have fun." The company was founded in 2014 and is headquartered in Paris, France. The target market is families, although it is exploring business applications as well. The company has won the 2016 Juniper Research Future digital award in the category 'Best Emerging & Disruptive Consumer Product'. It has developed the Buddy robot, which will be launched this year. Buddy is a companion robot developed to improve the everyday life of the user. He protects your home, offers assistance in the kitchen, entertains the family with music and videos, acts as a calendar and alarm clock, and interfaces with popular smart home solutions. Buddy is 56cm tall and weighs 5kg. His brain is an integrated android tablet. He is able to hear and speak thanks to built-in speakers and a microphone array system. He is fully mobile, with three wheels, and is equipped with a variety of sensors to deliver various functionalities. These sensors include a 3D camera, ultrasound, infra-red and thermal sensors, a range-finder sensor, and ground detectors. The sensors, coupled with Buddy's mobility, mean that he can travel autonomously and learn and interact with the world around him. Buddy is built on an open-source technology platform, and developers across the world can easily collaborate to build additional hardware and software for the device.



Interview with Rodolphe
Hasselvander, CEO Blue Frog
Robotics

Why did you start Blue Frog Robotics?

"I had a dream of a companion robot when I was younger and that dream is still very

much alive today. After developing robots for nuclear, healthcare, and defense applications, I believe the time for consumer robotics is now. With vacuum robots and drones in many homes, the general public is ready to accept a robot companion. Many people are already aware of how to use tablets and touch screens, so that's one of the reasons we built Buddy with this standard technology."

What were the goals when designing Buddy?

“When we set out to design Buddy we had two goals. First of all, we wanted the robot to be accepted by people. Therefore, it had to be cute, ‘Kawai’. Secondly, it had to be accessible, because we want to democratize robots. The challenge we faced was to create a cute robot that people want to bond with while using accessible technology.”

What markets are you targeting?

“The original idea of Buddy was born out of the need to help the elderly to live alone for longer. Many elderly people love our prototype. However, there was a risk that they would have the same concerns that many have about using a hearing aid: “Am I that old?” For this reason, we started testing Buddy in family units as well. Our thinking is that when elderly people see families adopting Buddy, it’ll become easier for them to do the same.”

the autism spectrum. We're working with 'Les Papillons Blancs' (an organization that helps those with intellectual disabilities). Also, there's a real need to bridge the gap between technology and the elderly, and Buddy plans to fulfill that role.

The value proposition also depends on the country. For example, in China, we'll focus on Buddy as a children's companion. There's a market in China of ~120 million households for a companion/education robot for children. In the United States, the focus will lie on home security, as that's a key concern for many people there."

What sets the Buddy platform apart?

"We use the Google app store. Our platform will be an open network to stimulate innovation. We're working with XBrain on AI.

Also, Buddy's an "open-hardware" platform
that can be expanded with accessories such as arms

How do you see Buddy evolving?

"Fundamentally, Buddy's about a new way of interacting with technology. A hub to integrate all the 'Internet of Things' devices in the home, perhaps. The elderly indicate, after interacting with Buddy, they want him to teach them new skills such as learning

“WE WANT TO
DEMOCRATIZE ROBOTS”

with a pico-projector and a side tray.”

What’s your business model?

“Our initial business model is to gain revenue by selling Buddy hardware and accessories. For the next phase, we aim to valorize the anonymized data Buddy gathers. We’re also exploring subscription-based

a new language or playing an instrument. We mentioned that there are already plenty of applications available on the computer or tablet, but they insisted that they wanted to learn through Buddy, because he can offer ‘character’, someone to “talk to”.

What’s Buddy’s value proposition?

“There’s Buddy’s ability to teach children through edutainment, for example. We’re currently working on educational applications. Also, we’ve received very positive reactions from those working with people who are on

models for services.”

What’s your perspective on AI?

“AI still has many challenges. It only works in highly structured and narrow spaces such as winning Jeopardy or doing diagnostic work. It’s not ready to truly socially interact. Artificial General Intelligence may not be with us before ~2050.”

What’s your goal?

“We want to put a robot in every home.”

AvatarMind was founded with the goal of creating intelligent humanoid robots to serve as helpers and companions. The company's dream is to create affordable robots that live in harmony with ordinary people and improve their everyday lives. The company was founded in 2014 and has facilities in Nanjing and Shenzhen, China and Silicon Valley in the US. Its first target market is Chinese families with children aged 3-8. It has developed the iPal robot, which will be launched later this year. The robot is a friendly companion for children to play with and talk to in a natural manner. It can dance, tell stories, play games, and enable children to chat with friends and family, share videos, and connect to social media. For parents, iPal is a teacher for their children, with spoken language learning and tablet-based educational programs. A key feature is that parents can use their smartphone to control iPal and monitor their child's safety and activity from anywhere in the world and at any time. iPal is 106.7cm tall and weighs 12.4kg. It has 25 motors, two cameras, five microphones, five touch sensors, four range finders, and seven obstacle detectors.



"IN ABOUT FIVE YEARS
OR SO PEOPLE WILL HAVE
MULTIPLE ROBOTS
IN THEIR HOME"

Can you tell us about the start of your product

– what unfulfilled need did you see?

“We started about two years ago after being in an incubation phase for a while. Our final prototype will be released in April, and we expect to begin manufacturing in the 3rd quarter of this year. Our first focus is the one-child market in China. We believe our robot can help mothers when they need a break, educate their children so they can prepare for school, make learning more fun, and enable them to monitor their safety and health.”

Can you tell us what iPal's value proposition is?

“iPal is designed to be a companion for young children and to help with their education. It'll also allow parents to monitor their children from the next room or communicate with them when they are on a business trip. We developed our own software so the robot can help to educate and entertain a child in a more interactive way. We're also looking for partners to work with in specific vertical applications such as elderly care and assisting children with special needs. In fact, we already have a partner we'll be working with to deliver robot therapies for autistic children.

iPal is a complex, but highly reliable and robust, device; it will still function when it gets knocked down, for example – this actually happened at the recent CES show, when a robot got knocked off a 3.5 foot high table.

We'll have our own app store with applications that are approved for children. As a result, the children will be safe when using applications developed for our robot.”

What metrics are you collecting in your deployments?

“A key part of our deployments is assessing the interaction with children. The child can talk to iPal and we need to make sure this interaction is optimal and fun. The robot should be able to hold the child's interest for a significant amount of time. This will be achieved by a combination of interesting interaction

behaviors, learning to adapt to the child's interests, and pushing new content to the robot every day. We have a large team currently working on this. For example, we're working with teachers to develop a long list of questions that children typically ask and age-appropriate answers."

What are your key market segments? How do you target these at the moment?

"We're focused on the Chinese market first. There's a very large market of one child families there. The key characteristic of these families is that they have disposable income and are known to support anything that will improve their child's quality of life, education, and security. We're also looking at several use-cases for the US. In addition to companion robots for young children, we've received a great deal of interest in iPal for elderly care and children with special needs (especially autism).

We intend to open 'robot playgrounds' in China. Shopping malls in China are like entertainment centers and there are often playgrounds for children. This is an ideal environment for us to get exposure and feedback. In our iPal robot playground stores in major malls, children will be able to play with iPal and parents can then buy iPal robots when they see their value. It offers parents telepresence and the ability to play games with their children while they're away. And no parent just wants his or her child to sit in front of a screen all day. As a robot that can move smoothly and elegantly, iPal will also help to promote physical activity and interactivity in children."

What's your perspective on social robots?

"At this stage of development, we aren't big believers in trying to build robots that look exactly like humans. We feel the fun, cartoonish form of iPal will be both attractive to children and non-threatening to adults. Our market surveys also support this. However, it's relatively easy and inexpensive to cover the robot core with different plastics, so different form factors for different markets are definitely possible.

We also want to strongly emphasize that we aren't trying to replace parents or friends. It's about filling gaps. Likewise with autism care.

Our typical use-case would be something like this: the therapist goes in, sets up the robot program. Ideally, therapeutic interactions with an autistic child can take place up to 30-40 hours a week. If you have a therapist that can use the robot as a surrogate, the effect can multiply. We're augmenting humans to make their work more effective so that more children can be helped by each therapist. We've a lot of interest from playground and kindergarten schools in China. They typically have one teacher to take care of 20-30 children. Again, imagine augmenting their capabilities with robots, so that each child will have more one on one learning, either with the teacher or with the robot as an interactive instructor."

Who are your key partners?

"We already have a good partner to explore the use of our robot for applications in autism. The humanoid form is great for that. Did you know there are 7 million autistic children in China? We're also actively looking for partners who are experts in other fields like elderly care and special needs children."

What are the key drivers for social robot adoption in your view?

"In China, the decision-maker is the head of the family. The key drivers are the ability to improve the child's quality of life and ensure his/her safety. We emphasize the educational aspect that will give a head start in school. About 85% of the children who took part in our test will come up to the robot right away



and play with it, but others sometimes take a little longer to warm to it.

The big lever is in areas where there is a clear utility. One of the things that has been very interesting and gratifying is that when we do trade shows, so many people come up with their own ideas on how to use iPal. As an example, physicians tell us that there are a lot of children in cancer wards who are lonely and unattended for much of the time. Imagine being able to improve their quality of life. A high-end hotel chain wants to talk with us about iPal leading their clients to their rooms and then explaining all the features of the room. It even looks like iPal will be starring in a children's TV show in the future."

How do you see the future of social robotics?

"The computer revolution is over – today everyone has multiple computers in their home. The next big revolution is going to be the robot revolution. In about five years or so people will have multiple robots in their home, not just their Roomba. However, it's also important to be realistic about what robots can do at this stage of the technology if you want to be successful in commercial markets. For example, we don't believe that the technology is good enough today to develop a convincing robot that can be a true companion for adults. However, we do believe that we can build great products for young children, and for specialized applications like elderly care and children with special needs."

Do contact the author for more information and/or send your email address so we can share updates as the social robot story evolves.

to learn more?

Contact

Roeland van Oers roeland@readyforrobots.com

Eric Wesselman wesselman.eric@kpmg.nl

[illegible]

LEA helps people to lead an active life, helps with daily routines, ensures safety and ease of communication

A robot that relays products from one person to another

A companion robot that protects the home, offers assistance, entertains and interfaces with smart home solutions

A friendly companion for children to play with and talk to in a natural manner

Helps adults to manage their life, helps seniors to live with greater independence and is a playmate for children

Your smart little companion

Create closer family relationships. A connection to home
devices and a companion for the family

A

“social dialogue” robot

Pepper is the first emotional robot. He was designed
to be a true companion for daily life

Your autonomous retail robot assistant.

Empowering your retail workforce with Inventory Management and Customer Service

A friendly full-size humanoid robot that will make your event unique

Entertainment at

Target market

industry

years

businesses

Elderly Care Hospitality

Families Children aged 3-8

Families Families Families Elderly care Both homes and

Retail

events, conferences and fairs

Company

Robot Care Systems

Savioke Blue Frog Robotics
Robots PAL Robotics

Avatarmind

Jibo

Asus

Future Robot

Vstone Aldebaran Fellow

Headquarters

The Netherlands USA France USA USA Taiwan Korea Japan France USA Spain

Website robotcaresystems. com

savioke.com

bluefrogrobotics.c

om

avatarmind.com

Jibo.com

zenbo.asus.com

myfuro.com

vstone.co.jp

aldebaran.com

fellowrobots.com

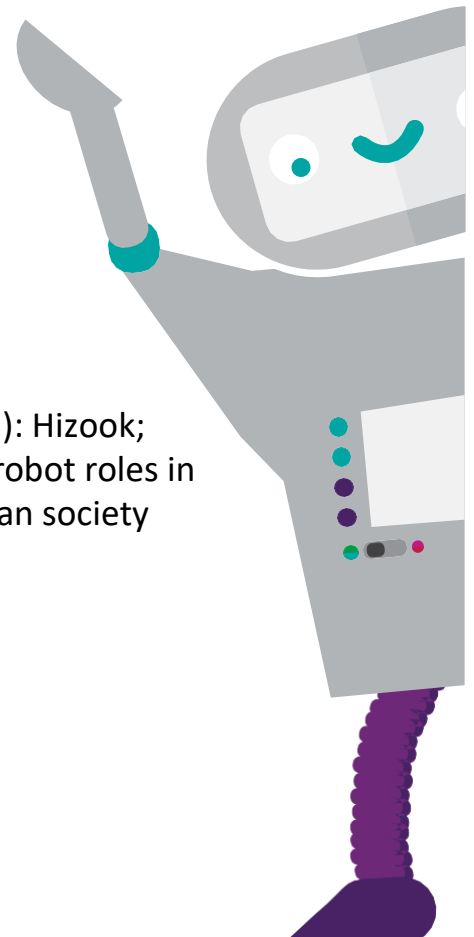
pal-robotics.com

Sources

- 1 <http://www.ifr.org/industrial-robots/statistics/>
- 2 <http://www.inc.com/graham-winfrey/introducing-pepper-the-friendly-humanoid-robot.html>
- 3 <http://www.techinsider.io/japan-developing-carebots-for-elderly-care-2015-11>
- 4 <https://ec.europa.eu/digital-single-market/node/376>
- 5 Robots at Work
(http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2575781)
- 6 https://en.wikipedia.org/wiki/Category:Deaths_caused_by_industrial_robots
- 7 <http://www.theguardian.com/world/2015/feb/09/south-korean-womans-hair-eaten-by-robot-vacuum-cleaner-as-she-slept>
- 8 <http://users.clas.ufl.edu/mcolburn/Web-links/Nursing%20Lecture/Why%20Don't%20People%20Who%20Need%20Them%20Get%20Hearing%20Aids.htm>
- 9 <http://www.popsoci.com/year-ahead-and-behind-robotics>
- 10 http://www.theregister.co.uk/2015/03/19/andrew_ng_baidu_ai/
- 11 <http://techcrunch.com/2013/07/29/why-labor-unions-and-silicon-valley-arent-friends-in-2-charts/>
- 12 <http://www.therobotreport.com/news/2013-stats-from-the-ifr>
- 13 <http://www.ifr.org/industrial-robots/statistics/>
- 14 <http://www.aesynt.com/iv-solutions/iv-station>
- 15 <http://www.amazon.com/Amazon-SK705DI-Echo/dp/B00X4WHP5E>
- 16 <http://www.mimitchi.com/aibo/>
- 17 <http://www.nytimes.com/2015/06/18/technology/robotica-sony-aibo-robotic-dog-mortality.html>
- 18 <http://www.nickbostrom.com/papers/survey.pdf>

Number of active robots: International Federation of Robotics, NASA, BI Intelligence
Personal robot sales 2015-2018 (million units): International Federation of Robotics
Interest in 'Companion Robot' over time: Google trends, extracted March 2016
Average sensor cost (\$): Goldman Sachs, BI Intelligence

Public venture capital robotics investments (\$ million): Hizook;
Robotics News for Academics & Professionals
Social robot roles in human society: Roles and functions of robots in human society (Dautenhahn, 2002)



The information contained herein is of a general nature and is not intended to address the circumstances of any particular individual or entity. Although we endeavor to provide accurate and timely information, there can be no guarantee that such information is accurate as of the date it is received or that it will continue to be accurate in the future. No one should act on such information without appropriate professional advice after a thorough examination of the particular situation.

© 2016 KPMG Advisory N.V., registered with the trade register in the Netherlands under number 33263682, is a member firm of the KPMG network of independent member firms affiliated with KPMG International Cooperative ('KPMG International'), a Swiss entity. All rights reserved. The name KPMG and

logo are registered trademarks of KPMG International.

7. References

- ABRAMSON, H., AND DAHL, V. (1989) *Logic grammars*. New York: Springer.
- AHO, A. V., AND LLMAN, J. D. (1972) *The theory of parsing, translation, and compiling*, vol. 1, *Parsing*. Englewood Cliffs, New Jersey: Prentice—Hall.
- AKMAJIAN, A.; DEMERS, R. A.; FARMER, A. K.; AND HARNISH, R. M. (1990) *Linguistics. no introduction to language and communication*. 3rd ed. Cambridge, Massachusetts: MIT Press.
- RAM, S. (1983) A two—level morphological analysis of Japanese. *Texas Linguistic Forum* 22:229—252.
- ALLEN, J. F. (1983) Maintaining knowledge about temporal intervals. *Communications of the ACM* 26:832—843. Reprinted in BRACHMAN AND LEVESQUE (1985) 509—521.
- ALLEN, J. F. (1987) *Natural language understanding*. Menlo Park, California: Benjamin—Cummings.
- ANDERSON, S. R. (1988a) Morphology as a parsing problem. In WARLACE (1988) 4—21. Also published in *Linguistics* 26:521—544.
- ANDERSON, S. R. (1988b) Morphological theory. In NEWMAYER (1988, vol. 1) 146—191.
- BACH, E. W. (1989) *Informal lectures on formal semantics*. Albany, New York: State University of New York Press.
- BARTON, G. E. (1985) On the complexity of ID/LP parsing. *Computational Linguistics*

- BARTSCH, R. (1987) Context—dependent interpretation of lexical items. In J.; DE JONGH, D.; AND STOKHOF, M., eds., *Foundations of Pragmatics and Semantics* (Groningen-Amsterdam Studies in Semantics, 7), 1—26. Dordrecht: Foris.
- BARWISE, J., AND COOPER, R. (1981) Generalized quantifiers and natural language. *linguistics and philosophy* 4:159—219. Reprinted in KurAS ET Ac. (1988), 241—301.
- BARWISE, J., AND ETCEHEMENDY, J. (1991) *The language of first-order logic*. (CSLI Lecture Notes, 23.) 2nd ed. Stanford: Center for the Study of Language and Information.
- BATES, M. (1978) The theory and practice of augmented transition network grammars. In BoLC (1978) 191—259.
- BERWICK, R. C.; ABNEY, S. P.; AND NNY, C., EDS. (1991) *Principle -based parsing: compiitation and psycholinguistics*. Dordrecht: Kluwer.
- BLOOMFIELD, L. (1933) *Language*. New York: Holt, Rinehart, Winston. Reprinted 1984, Chicago: University of Chicago Press.
- BLOOMFIELD, L. (1939) Menomini morphophonemics. *Travaux du cercle linguistique de Prague* 8:105—115. Reprinted in MAKKAI, V. B., ED., *Phonological theory*, 58—64. New York: Holt, Rinehart, Winston, 1972.
- BLUM, B. I. (1966) Free—text inputs to utility routines. *C mmlfnicalions Of the ACM* 9:525—526.
- BOBROW, D. G. (1968) Natural language input for a computer problem—solving system. In MINSKY (1968) 135—215.
- BoiSEN, S. (1988) Pro—KIMMO: a Prolog implementation of two—level morphology. In WALLACE (1988) 31—53.
- BOLC, L., ed. (1978) *Natural language communication with computers*. (Lecture Notes in Computer Science, 63.) Berlin: Springer.
- BRACHMAN, R. J., AND LEVESQUE, H. J. (1985) *Readings in knowledge representation*. Los Altos, California: Kaufmann.
- BRESNAN, J., ed. *The mental representation of grammatical relations*. Cambridge, Massachusetts: MIT Press.

BUCHMANN, B. (1987) Early history of machine translation. In KING (1987) 3—21. CARDEN, G. (1986) Blocked forwards coreference, unblocked forwards anaphora: evidence for an abstract model of coreference. *CLS* 22, Part 1, 262—276. Chicago Linguistic Society.

CHARNIAK, E. (1973) Jack and Janet in search of a theory of knowledge. *Advance papers from the Third International joint Conference on Artificial Intelligence [IJCAI]*, 337—

343. Los Altos, California: Kaufmann. Reprinted in GROSZ ET AL. (1986) 331—337.

CHIERCHIA, G. (1982) Bare plurals, mass nouns, and nominalization. *Proceedings, First West Coast Conference on Formal Linguistics (WCCFL I)*, 243—255. Stanford University.

CHIERCHIA, G., AND McCONNELL—GINET, S. (1990) *Meaning and grammar. an introduction to semantics*. Cambridge, Massachusetts: MIT Press.

HOMSKY, N. (1957) *Syntactic structures*. The Hague: Mouton.

- HOMSKY, N. (1965) *Aspects of the theory of syntax*. Cambridge, Massachusetts: MIT Press.
- CHOMSKY, N. (1982) *Lectures on government and binding*. Dordrecht: Foris.
- CHOMSKY, N., AND HALLE, M. (1968) *The sound pattern of English*. New York: Harper and Row.
- CHURCH, A. (1941) *The calculi of lambda-conversion*. Princeton: Princeton University Press.
- COLMERAUER, A. (1978) Metamorphosis grammars. In Borc (1978) 133—189. COOK, V. J. *Chomsky's universal grammar: an introduction*. Oxford: Blackwell. COOPER, R. (1983) *Quantification and syntactic theory*. Dordrecht: Reidel.
- COTTRELL, G. W. (1989) *A connectionist approach to word sense disambiguation*. London: Pitman.
- COVINGTON, M. (1984) *Syntactic theory in the High Middle Ages*. Cambridge, England: Cambridge University Press.
- COVINGTON, M. (1989) *GULP 2.0: an extension of Prolog for unification-based grammar*. Research Report AI—1989-01, Artificial Intelligence Programs, The University of Georgia.
- COVINGTON, M. (1990) Parsing discontinuous constituents in dependency grammar. *Computational Linguistics* 16:234—236.
- COVINGTON, M.; NUTE, D.; GOODMAN, D.; AND SCHMITZ, N. (1988) *From English to Prolog via discourse representation theory*. Research Report 01-0024, Artificial Intelligence Programs, The University of Georgia.
- COVINGTON, M.; NUTE, D.; AND ELLINO, A. (1988) *Prolog programming in depth*. Glenview, Illinois: Scott, Foresman. (Second edition in preparation, to be published by Prentice—Hall.)
- CRYSTAL, D. (1987) *The Cambridge encyclopedia of language*. Cambridge, England:

Cambridge University Press.

CULLINGFORD, R. (1981) SAM and Micro—SAM. In SCHANK AND RIESBECK (1981) 75—
135.

DAHL, V., AND SAINT-DIZIER, P., eds. (1985) *Natural-language understanding and logic programming*. Amsterdam: North-Holland.

DAHL, V., AND SAINT-DIZIER, P., eds. (1988) *Natural-language Understanding and logic programming, II*. Amsterdam: North—Holland.

DATE, C. J. (1990) *An introduction to database systems*, vol. 1. 5th ed. Reading, Massachusetts: Addison-Wesley.

DE LA BRIANDAIS, R. (1959) File searching by using variable length keys. *Proceedings of the Western Joint Computer Conference* 15:295—298. New York: Institute of Radio Engineers.

DENES, P. B., AND PINSON, E. N. (1963) *The speech chain: the physics and biology of spoken language*. Bell Telephone Laboratories.

DOWTY, D. R. (1979) *Word meaning and Montague grammar*. Dordrecht: Reidel. DOWTY, D. R. (1989) On the semantic content of the notion 'thematic role.' In CHIERS-

CHIA, G.; PARTEE, B. H.; AND RNER, R., eds., *Properties, types, and meaning*, vol.

- DYER, M. G. (1983) *In-depth understanding: a computer model of integrated processing for narrative comprehension*. Cambridge, Massachusetts: MIT Press.
- EARLEY, J. (1970) An efficient context—free parsing algorithm. *Communications of the ACM* 13:94—102. Reprinted in GRO SZ ET AL. (1986) 25—33.
- EMONDS, J. E. (1976) *A transformational approach to English syntax*. New York: Academic Press.
- ESTIVAL, D. (1990a) Generating French with a reversible unification grammar. *Proceedings of the 13th International Conference on Computational Linguistics (COLING 90)*, 106—112.
- ESTIVAL, D. (1990b) ELU: an environment for machine translation (project note). *Proceedings of the 13th International Conference on Computational Linguistics [COLING 90]*, 385—387.
- EVENS, M. W., ed. (1988) *Relational models of the lexicon: representing knowledge in semantic networks*. Cambridge, England: Cambridge University Press.
- FILLMORE, C. (1968) The case for case. In BACH, E., AND HARMS, R. T., EDS., *Universals in linguistic theory*, 1—88. New York: Holt, Rinehart, and Winston.
- FLECK, M. M. (1988) *Boundaries and topological algorithms*. Technical Report 1065, MIT Artificial Intelligence Laboratory.
- FREDKIN, E. (1960) Trie memory. *Communications of the ACM* 3:490—499.
- FROMKIN, J., AND RODMAN, R. (1988) *An introduction to language*. 4th ed. New York: Holt, Rinehart, and Winston.
- GAL, A.; APALME, G.; SAINT-DIZIER, P.; AND SOMERS, H. (1991) *Prolog for natural language processing*. Chichester, England: Wiley.
- GAZDAR, G., FRANZ, A.; OSBORNE, K.; AND EVANS, R. *Natural language processing in the 1980s: a bibliography*. (CSLI Lecture Notes, 12.) Stanford: Center for the Study of Language and Information.
- GAZDAR, G., KLEIN, E.; PULLUM, G.; AND SAG, I. (1985) *Generalized phrase structure grammar*. Cambridge, Massachusetts: Harvard University Press.
- GAZDAR, G., AND MELLISH, C. (1989) *Natural language processing in Prolog*. *rim intro-*

duction to computational linguistics. Wokingham: Addison—Wesley.

GERDEMANN, D. (1989) Restriction as a means of optimizing unification parsing. *Studies in the Linguistic Sciences* 19.1:81—92.

GERDEMANN, D. (1991) *Parsing and generation of unification grammars*. Cognitive science technical report CS—91—06, The Beckman Institute, Urbana, Illinois.

GRICE, H. P. (1975) Logic and conversation. CorE, P. AND J. MORGAN, EDS., *byn/nr and semantics*, vol. 3, 41—58. New York: Academic Press.

GROSS, M. (1979) On the failure of generative grammar. *Language* 55:859—885.

GROSZ, B. J.; SPARCK JONES, K.; AND WEBBER, B. L. (1986) *Readings in natural language processing*. Los Altos, California: Kaufmann.

GRUBER, J. S. (1965) *Studies in lexical relations*. Thesis, Ph.D., Massachusetts Institute of Technology. Published as Part 1 of GRUBER (1976).

- GRUBER, J. S. (1976) *Lexical structures in syntax and semantics*. Amsterdam: North—Holland.
- HALLE, M. (1959) *The sound pattern of Russian*. The Hague: Mouton.
- HERZOG, O., AND RORRINGER, C. R., eds. (1991) *Text understanding in LILOG: tritegrating computational !*8*!•*•'• sOnd cirtificial intelligence*. (Lecture Notes in Artificial Intelligence, 546.) Berlin: Springer.
- HIRST, G. (1987) *Semantic interpretation and the resolution of ambiguity*. Cambridge: Cambridge University Press.
- HOBBS, J. R. (1978) Resolving pronoun references. *Lingua* 44:311-388. Reprinted in GROSZ ET AL. (1986) 339—352.
- HOBBS, J. R., AND HIEBER, S. M. (1987) An algorithm for generating quantifier scopings. *Computational Linguistics* 13:47—63.
- HORN, L. R. (1989) *A natural history of negation*. Chicago: University of Chicago Press.
- HORNBY, A. S. (1989) *Oxford advanced learner's dictionary of current English*. 4th ed. Oxford University Press.
- HORNSTEIN, H. (1984) *Logic as grammar: an approach to meaning in natural language*. Cambridge, Massachusetts: MIT Press.
- HUDDLESTON, R. (1988) *English grammar. ari outline*. Cambridge, England: Cambridge University Press.
- HUDSON, R. A. (1988) Coordination and grammatical relations. *Journal of Linguistics* 24:303—342.
- HUTCHINS, . J. (1986) *Machine translation. past, present, future*. Chichester: Ellis Hor—wood.
- IRONS, E. T. (1961) A syntax directed compiler for ArGO 60. *Communications of the*

- ACM 4:51—55.
- ISO. See: ScOWEN (1992).
- JACKENDOFF, R. S. (1972) *Semantic interpretation in generative grammar*. Cambridge, Massachusetts: MIT Press.
- JACKENDOFF, R. S. (1977) *X syntax: a study of phrase structure*. Cambridge, Massachusetts: MIT Press.
- JESPERSEN, O. (1949) *A modern English grammar on historical principles*, part VII: *Syntax*—Lux (completed by NIELS HAISSUND). Copenhagen: Munksgaard.
- JOHNSON, M. (1991) Features and formulae. *Computational Linguistics* 17:131—151.
- JOHNSON, M., AND KLEIN, E. (1986) *Discourse, anaphora, and parsing*. Report No. CSLI-86-63. Center for the Study of Language and Information, Stanford University. Also in *Proceedings of the 11th International Conference on Computational Linguistics (COLING 86)*, 669—675.
- KAC, M. (1982) Review of Marcus (1980). *Language* 58:447—455.
- KAMP, H. (1981) A theory of truth and semantic representation. In GROENENDIJK, J., ET AL. (eds.) *Formal methods in the study of language*, 277—322. University of Amsterdam. Reprinted in GROENENDIJK, J., ET AL. (eds.), *Truth, interpretation, and information*.

- and Semantics*, vol. 7: *Notes from the linguistic underground*, 363—385. New Academic Press.
- KARTTUNEN, L. (1983) KIMMO: a general morphological processor. *Texas Linguistic Forum* 22:165—186.
- KARTTUNEN, L., AND BY, M. (1985) Parsing in a free word order language. In DowT ET AL. (1985) 279—306.
- KARTTUNEN, L., AND ITTENBURG, K. (1983) A two—level morphological analysis of English. *Texas Linguistic Forum* 22:217—227.
- KASHKET, M. (1991) Parsing Warlpiri, a free-word-order language. In BERWICK ET AL. (1991) 123—151.
- ITS, M. (1980) *Algorithm schemata and data structures in syntactic processing*. Research report, Xerox Palo Alto Research Center. Reprinted in GROSZ ET AL. (1986) 35—70.
- KAY, M. (1985) Parsing in functional unification grammar. In DowTY ET AL. (1985) 251—278.
- KHAN, R. (1983) A two-level morphological analysis of Rumanian. *Texas Linguistic Forum* 22:253—270.
- LBURY, J. (1984) *Earle y-basierte Algorithmen fur direktes Parsen mit ID7LP-Grammatiken*. KIT—Report 16, Institut fiir angewandte Informatik, Technische Universitiit Berlin.
- LNG, M., ed. (1983) *Parsing natural language*. London: Academic Press.
- KING, M., ed. (1987) *Machine translation today. the state of the art*. Edinburgh: Edinburgh University Press.
- KNUTH, D. E. (1973) *The art of computer programming*, vol. 3: *Sorting and searching*. Reading, Massachusetts: Addison—Wesley.
- KOLODNER, J. L. (1984) *Retrieval and organizational strategies in conceptual memory: a computer model*. Hillsdale, New 'Jersey: Erlbaum.
- KOSKENNIEMI, K. (*9*3) *Two-level morphology: a general computational model for word- form recognition and production*. Publication No. 11, Department of General Linguistics, University of Helsinki.
- KULAS, J.; FETZER, J. H.; AND RANKIN, T. L., eds. (1988) *Philosophy, language, and*

artificial intelligence. Dordrecht: Kluwer.

LANE, A. (1987) DOS in English. *Byte* 12.14 (December 1987) 261—264.

LEHNERT, W. G. (1988) Knowledge-based natural language understanding. In SHROBE,

H. E., ED., *Exploring artificial intelligence: survey talks from the National Conferences on Artificial Intelligence*, 83—131. San Mateo, California: Kaufmann.

LEISS, H. (1990) On Kilbury's modification of Earley's algorithm. *ACM Transactions on Programming Languages and Systems* 12:610—640.

LEVINSON, S. C. (1983) *Pragmatics*. Cambridge, England: Cambridge University Press. LUN, S.

(1983) A two—level morphological analysis of French. *Texas Linguistic Forum*

22:271—278.

MARCUS, M. P. (1978) A computational account of some constraints on language. In WALTZ, D., ED., *Theoretical issues in natural language processing—2* (Urbana, Illinois: Association for Computational Linguistics), 236—246. Reprinted in GROsz ET AL.

(1986) 89—99.

- MARCUS, M. P. (1980) *A theory of syntactic recognition for natural language*. Cambridge, Massachusetts: MIT Press.
- MATSUMOTO, Y.; TANAKA, H.; HIRAKAWA, H.; MIYOSHI, H.; AND YASUKAWA, H. (1983) BUP: a bottom—up parser embedded in Prolog. *New Generation Computing* 1:145—158.
- MATTHEWS, P. H. (1991) *Morphology*. 2nd edition. Cambridge, England: Cambridge University Press.
- MATTHEWS, P. H. (1981) *Syntax*. Cambridge, England: Cambridge University Press.
- MCCAWLEY, J. D. (1981) *Everything that linguists have n/ways wanted to know about logic*. Chicago: University of Chicago Press.
- MINSKY, M., ed. (1968) *Semantic information processing*. Cambridge, Massachusetts: MIT Press.
- MONTAGUE, R. (1973) The proper treatment of quantification in ordinary English. In HINTIKKA, J., ET AL., eds., *Approaches to Natural Language*, 221—242, Dordrecht: Reidel. Reprinted in THOMASON, R. H., ED., *Formal philosophy: selected papers of Richard Montague*, 247—270, New Haven: Yale University Press, 1974. Also reprinted in KELAS ET AL. (1988), 141—162.
- MOYNE, J. A. (1985) *Understanding language: man or machine*. New York: Plenum.
- NAISH, L. (1986) *Negation and control in Prolog*. (Lecture Notes in Computer Science, 238.) Berlin: Springer.
- NEWMAYER, F. J. (1983) *Grammatical theory: its limits and possibilities*. Chicago: University of Chicago Press.
- NEWMAYER, F. J. (1986) *Linguistic theory in America*. 2nd edition. Orlando: Academic Press.
- NEWMAYER, F. J., ed. (1988) *Linguistics. the Cambridge survey*. 4 vols. Cambridge, England: Cambridge University Press.
- NILSSON, U. (1986) AID: an alternative implementation of DCGs. *New Generation Computing* 4:383—399.
- NIRENBURG, ., ED. (1987) *Machine translation. theoretical and methodological issues*. Cambridge, England: Cambridge University Press.
- NORVIG, P. (1991) *Paradigms of artificial intelligence programming: Case studies in Common Lisp*. San Mateo, California: Kaufmann.
- NUTE, D. (1988) Defeasible reasoning: a philosophical analysis in Prolog. In FETZER, J. H., ed., *Aspects of artificial intelligence* 251—288. Dordrecht: Kluwer.
- OBERMEIER, K. K. (1989) *Natural language processing technologies in artificial intelli-*

gence: the science and industry perspective. Chichester: Ellis Horwood.

OJEDA, A. E. (1991) Definite descriptions and definite generics. *Linguistics and Philosophy* 14:367—397.

O'KEEFE, R. A. (1985) On the treatment of cuts in Prolog source—level tools. *Proceedings, 1985 Symposium on Logic Programming*, 68—72. IEEE Computer Society.

O'KEEFE, R. A. (1990) *The craft of Prolog*. Cambridge, Massachusetts: MIT Press.

OKuNisiii, T.; SUGIMURA, R.; AND MATSUMOTO, Y. (1988) Comparison of logic programming based natural language parsing systems. In DAHL, V., AND SAINT-DIZIER, P., eds., *Natural language understanding and logic programming, II*, 1—14. Amsterdam:

A. Andreevskaia and S. Bergler, Mining WordNet for a fuzzy sentiment: Sentiment tag extraction from WordNet glosses, *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*, Trento, Italy, 2006.

N. Archak, A. Ghose, and P. Ipeirotis, Show me the money! Deriving the pricing power of product features by mining consumer reviews, *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, San Jose, CA, 2007.

A. Aue and M. Gamon, Customizing sentiment classifiers to new domains: A case study, *Proceedings of Recent Advances in Natural Language Processing (RANLP)*, Borovets, Bulgaria, 2005.

P. Beineke, T. Hastie, C. Manning, and S. Vaithyanathan, Exploring sentiment summarization, *Proceedings of the AAAI Spring Symposium on Exploring Attitude and Affect in Text*, Stanford, CA, AAAI Technical Report SS-04-07, 2004.

S. Bethard, H. Yu, A. Thornton, V. Hatzivassiloglou, and D. Jurafsky, Automatic extraction of opinion propositions and their holders, *Proceedings of the AAAI Spring Symposium on Exploring Attitude and Affect in Text*, Stanford, CA, 2004.

J. Blitzer, M. Dredze, and F. Pereira, Biographies, Bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification, *Proceedings of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic, 2007.

J. Bos and M. Nissim, An empirical approach to the interpretation of superlatives, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Sydney, Australia, 2006.

E. Breck, Y. Choi, and C. Cardie, Identifying expressions of opinion in context, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Hyderabad, India, 2007.

G. Carenini, R. Ng, and A. Pauls, Multi-document summarization of evaluative text, *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*, Trento, Italy, pp. 305—312, 2006.

- G. Carenini, R. T. Ng, and E. Zwart, Extracting knowledge from evaluative text, *Proceedings of International Conference on Knowledge Capture (K-CAP)*, Banff, Canada, pp. 11–18, 2005.
- Y. Chen and J. Xie, Online consumer review: Word-of-mouth as a new element of marketing communication mix, *Management Science*, 54, 477–491, 2008.
- P. Chesley, B. Vincent, L. Xu, and R. Srihari, Using verbs and adjectives to automatically classify blog sentiment, *AAAI Symposium on Computational Approaches to Analysing Weblogs (AAAI-CAAW)*, Stanford, CA, pp. 27–29, 2006.
- Y. Choi, E. Breck, and C. Cardie, Joint extraction of entities and relations for opinion recognition, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Sydney, Australia, 2006.
- Y. Choi, C. Cardie, E. Riloff, and S. Patwardhan, Identifying sources of opinions with conditional random fields and extraction patterns, *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, Vancouver, Canada, 2005.
- S. R. Das and M. Y. Chen, Yahoo! for Amazon: Sentiment extraction from small talk on the Web, *Management Science*, 53, 1375–1388, 2007.
- K. Dave, S. Lawrence, and D. M. Pennock, Mining the peanut gallery: Opinion extraction and semantic classification of product reviews, *Proceedings of WWW*, Budapest, Hungary, pp. 519–528, 2003.
- C. Dellarocas, X. Zhang, and N. F. Awad, Exploring the value of online product ratings in revenue forecasting: The case of motion pictures, *Journal of Interactive Marketing*, 21, 23–45, 2007.
- A. Devitt and K. Ahmad, Sentiment analysis in financial news: A cohesion based approach, *Proceedings of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic, pp. 984–991, 2007.
- X. Ding, B. Liu, and P. S. Yu, A holistic lexicon-based approach to opinion mining, *Proceedings of the Conference on Web Search and Web Data Mining (WSDM)*, Stanford, CA, 2008.
- K. Eguchi and V. Lavrenko, Sentiment retrieval using generative models, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Sydney, Australia, pp. 345–354, 2006.
- A. Esuli and F. Sebastiani, Determining the semantic orientation of terms through gloss analysis, *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, Bremen, Germany, 2005.
- A. Esuli and F. Sebastiani, Determining term subjectivity and term orientation for opinion mining, *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*, Sydney, Australia, 2006.
- A. Esuli and F. Sebastiani, SentiWordNet: A publicly available lexical resource for opinion mining, *Proceedings of Language Resources and Evaluation (LREC)*, Genoa, Italy, 2006.
- A. Esuli and F. Sebastiani, PageRanking WordNet synsets: An application to opinion mining, *Proceedings of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic, 2007.
- C. Fellbaum, ed., *Wordnet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.
- M. Fiszman, D. Demner-Fushman, F. Lang, P. Goetz, and T. Rindfleisch, Interpreting comparative constructions in biomedical text, *BioNLP*, Prague, Czech Republic, 2007.

- M. Gamon, Sentiment classification on customer feedback data: Noisy data, large feature vectors, and the role of linguistic analysis, *Proceedings of the International Conference on Computational Linguistics (COLING)*, Geneva, Switzerland, 2004.
- M. Gamon, A. Aue, S. Corston-Oliver, and E. Ringger, Pulse: Mining customer opinions from free text, *Proceedings of the International Symposium on Intelligent Data Analysis (IDA)*, Madrid, Spain, pp. 121–132, 2005.
- G. Ganapathibhotla and B. Liu, Identifying preferred entities in comparative sentences, *Proceedings of the International Conference on Computational Linguistics, COLING*, Manchester, U.K., 2008.
- R. Ghani, K. Probst, Y. Liu, M. Krema, and A. Fano, Text mining for product attribute extraction, *SIGKDD Explorations Newsletter*, 8, 41–48, 2006.

- A. Ghose and P. G. Ipeirotis, Designing novel review ranking systems: Predicting usefulness and impact of reviews, *Proceedings of the International Conference on Electronic Commerce (ICEC)*, Minneapolis, MN, 2007.
- A. Ghose, P. G. Ipeirotis, and A. Sundararajan, Opinion mining using econometrics: A case study on reputation systems, *Proceedings of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic, 2007.
- N. Godbole, M. Srinivasaiah, and S. Skiena, Large-scale sentiment analysis for news and blogs, *Proceedings of the International Conference on Weblogs and Social Media (ICWSM)*, Boulder, CO, 2007.
- V. Hatzivassiloglou and K. McKeown, Predicting the semantic orientation of adjectives, *Proceedings of the Joint ACL/EACL Conference*, Toulouse, France, pp. 174–181, 1997.
- V. Hatzivassiloglou and J. Wiebe, Effects of adjective orientation and gradability on sentence subjectivity, *Proceedings of the International Conference on Computational Linguistics (COLING)*, Saarbrücken, Germany, 2000.
- M. Hu and B. Liu, Mining and summarizing customer reviews, *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, Seattle, WA, pp. 168–177, 2004.
- N. Hu, P. A. Pavlou, and J. Zhang, Can online reviews reveal a product's true quality?: Empirical findings and analytical modeling of online word-of-mouth communication, *Proceedings of Electronic Commerce (EC)*, Ann Arbor, MI, pp. 324–330. ACM, New York, 2006.
- N. Jindal and B. Liu, Identifying comparative sentences in text documents, *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR)*, Seattle, WA, 2006a.
- N. Jindal and B. Liu, Mining comparative sentences and relations, *Proceedings of AAAI*, Boston, MA, 2006b.
- N. Jindal and B. Liu, Review spam detection, *Proceedings of WWW*, Banff, AB, 2007. (Poster paper).
- N. Jindal and B. Liu, Opinion spam and analysis, *Proceedings of the Conference on Web Search and Web Data Mining (WSDM)*, Stanford, CA, pp. 219–230, 2008.
- N. Kaji and M. Kitsuregawa, Building lexicon for sentiment analysis from massive collection of HTML documents, *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, Prague, Czech Republic, pp. 1075–1083, 2007.
- J. Kamps, M. Marx, R. J. Mokken, and M. de Rijke, Using WordNet to measure semantic orientation of adjectives, *Proceedings of the LREC'04*, Lisbon, Portugal, pp. 1115–1118, 2004.
- H. Kanayama and T. Nasukawa, Fully automatic lexicon expansion for domain-oriented sentiment analysis, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Sydney, Australia, pp. 355–363, July 2006.
- A. Kennedy and D. Inkpen, Sentiment classification of movie reviews using contextual valence shifters, *Computational Intelligence*, 22, 110–125, 2006.
- S.-M. Kim and E. Hovy, Determining the sentiment of opinions, *Proceedings of the International Conference on Computational Linguistics (COLING)*, Geneva, Switzerland, 2004.
- S.-M. Kim and E. Hovy, Automatic identification of pro and con reasons in online reviews, *Proceedings of the COLING/ACL Main Conference Poster Sessions*, Sydney, Australia, pp. 483–490, 2006.
- S.-M. Kim and E. Hovy, Crystal: Analyzing predictive opinions on the web, *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL)*, Prague, Czech Republic, 2007.

- S.-M. Kim, P. Pantel, T. Chklovski, and M. Pennacchiotti, Automatically assessing review helpfulness, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Sydney, Australia, pp. 423–430, 2006.
- N. Kobayashi, K. Inui, and Y. Matsumoto, Extracting aspect-evaluation and aspect-of relations in opinion mining, *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Prague, Czech Republic, pp. 1065–1074, 2007.

- N. Kobayashi, K. Inui, Y. Matsumoto, K. Tateishi, and T. Fukushima, Collecting evaluative expressions for opinion extraction, *Proceedings of the First International Joint Conference on Natural Language Processing (IJCNLP)*, Hainan Island, China, pp. 584–589, 2004.
- L.-W. Ku, Y.-T. Liang, and H.-H. Chen, Opinion extraction, summarization and tracking in news and blog corpora, *AAAI Symposium on Computational Approaches to Analysing Weblogs (AAAI-CAAW)*, Stanford, CA, pp. 100–107, 2006.
- L.-W. Ku, Y.-T. Liang, and H.-H. Chen, Novel relationship discovery using opinions mined from the web, *National Conference on Artificial Intelligence (AAAI)*, Boston, MA, 2006.
- J. Lafferty, A. McCallum, and F. Pereira, Conditional random fields: Probabilistic models for segmenting and labeling sequence data, *Proceedings of ICML*, Williamstown, MA, pp. 282–289, 2001.
- B. Liu, *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer-Verlag, Berlin, Germany, 2006.
- B. Liu, M. Hu, and J. Cheng, Opinion observer: Analyzing and comparing opinions on the web, *Proceedings of WWW*, Chiba, Japan, 2005.
- J. Liu, Y. Cao, C.-Y. Lin, Y. Huang, and M. Zhou, Low-quality product review detection in opinion summarization, *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, Prague, Czech Republic, pp. 334–342, 2007. (Poster paper).
- Y. Liu, J. Huang, A. An, and X. Yu, ARSA: A sentiment-aware model for predicting sales performance using blogs, *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR)*, Amsterdam, the Netherlands, 2007.
- Y. Lu and C. X. Zhai, Opinion integration through semi-supervised topic modeling, *Proceedings of 2008 International WWW Conference (WWW'08)*, Beijing, China, pp. 121–130, 2008.
- C. MacDonald, I. Ounis, and I. Soboroff, Overview of the TREC2007 Blog Track, 2007. <http://trec.nist.gov/pubs/trec16/papers/BLOG.OVERVIEW16.pdf>
- J. Martin and P. White, *The Language of Evaluation, Appraisal in English*. Palgrave Macmillan, London/New York, 2005.
- R. McDonald, K. Hannan, T. Neylon, M. Wells, and J. Reynar, Structured models for fine-to-coarse sentiment analysis, *Proceedings of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic, pp. 432–439, 2007.
- Q. Mei, X. Ling, M. Wondra, H. Su, and C. X. Zhai, Topic sentiment mixture: Modeling facets and opinions in weblogs, *Proceedings of WWW*, Banff, Canada, pp. 171–180, 2007.
- R. Mihalcea, C. Banea, and J. Wiebe, Learning multilingual subjective language via cross-lingual projections, *Proceedings of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic, pp. 976–983, 2007.
- S. Morinaga, K. Yamanishi, K. Tateishi, and T. Fukushima, Mining product reputations on the Web, *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, Edmonton, Canada, pp. 341–349, 2002. (Industry track).
- T. Nasukawa and J. Yi, Sentiment analysis: Capturing favorability using natural language processing, *Proceedings of the Conference on Knowledge Capture (K-CAP)*, Sanibel Island, FL, 2003.

- V. Ng, S. Dasgupta, and S. M. N. Arifin, Examining the role of linguistic knowledge sources in the automatic identification and classification of reviews, *Proceedings of the COLING/ACL Main Conference Poster Sessions*, Sydney, Australia, pp. 611–618, 2006.
- X. Ni, G.-R. Xue, X. Ling, Y. Yu, and Q. Yang, Exploring in the weblog space by detecting informative and affective articles, *Proceedings of WWW*, Banff, Canada, 2007.
- K. Nigam and M. Hurst, Towards a robust metric of polarity, in *Computing Attitude and Affect in Text: Theories and Applications*, Vol. 20, *The Information Retrieval Series*, J. G. Shanahan, Y. Qu, and J. Wiebe, (eds.), Springer-Verlag, Berlin, Germany, 2006.

- B. Pang and L. Lee, A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts, *Proceedings of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic, pp. 271–278, 2004.
- B. Pang and L. Lee, Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales, *Proceedings of the Association for Computational Linguistics (ACL)*, Ann Arbor, MI, pp. 115–124, 2005.
- B. Pang and L. Lee, Opinion mining and sentiment analysis, *Foundations and Trends in Information Retrieval*, 2(1–2), 1–135, 2008.
- B. Pang, L. Lee, and S. Vaithyanathan, Thumbs up? Sentiment classification using machine learning techniques, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Philadelphia, PA, pp. 79–86, 2002.
- D.-H. Park, J. Lee, and I. Han, The effect of on-line consumer reviews on consumer purchasing intention: The moderating role of involvement, *International Journal of Electronic Commerce*, 11, 125–148, 2007.
- W. Parrott, *Emotions in Social Psychology*. Psychology Press, Philadelphia, PA, 2001.
- L. Polanyi and A. Zaenen, Contextual lexical valence shifters, *Proceedings of the AAAI Spring Symposium on Exploring Attitude and Affect in Text*, Stanford, CA, 2004.
- A.-M. Popescu and O. Etzioni, Extracting product features and opinions from reviews, *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, Vancouver, Canada, 2005.
- G. Qiu, B. Liu, J. Bu, and C. Chen, Expanding domain sentiment lexicon through double propagation, *International Joint Conference on Artificial Intelligence (IJCAI-09)*, Pasadena, CA, 2009.
- E. Riloff, S. Patwardhan, and J. Wiebe, Feature subsumption for opinion analysis, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Sydney, Australia, 2006.
- E. Riloff and J. Wiebe, Learning extraction patterns for subjective expressions, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Sapporo, Japan, 2003.
- E. Riloff, J. Wiebe, and T. Wilson, Learning subjective nouns using extraction pattern bootstrapping, *Proceedings of the Conference on Natural Language Learning (CoNLL)*, Edmonton, Canada, pp. 25–32, 2003.
- S. Sarawagi, Information extraction, *Foundations and Trends in Databases*, 1(3), 261–377, 2008.
- Y. Seki, K. Eguchi, N. Kando, and M. Aono, Opinion-focused summarization and its analysis at DUC 2006, *Proceedings of the Document Understanding Conference (DUC)*, New York, pp. 122–130, 2006.
- B. Snyder and R. Barzilay, Multiple aspect ranking using the good grief algorithm, *Proceedings of Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, HLT-NAACL 2007*, Rochester, NY, pp. 300–307.
- X. Song, Y. Chi, K. Hino, and B. Tseng, Identifying opinion leaders in the blogosphere, *Proceedings of the Conference on Information and Knowledge Management (CIKM)*, Lisbon, Portugal, pp. 971–974, 2007.
- A. Stepinski and V. Mittal, A fact/opinion classifier for news articles, *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR)*, Amsterdam, the Netherlands, pp. 807–808, 2007.
- P. J. Stone. *The General Inquirer: A Computer Approach to Content Analysis*. MIT Press, Cambridge,

MA, 1966.

- V. Stoyanov and C. Cardie, Partially supervised coreference resolution for opinion summarization through structured rule learning, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Sydney, Australia, pp. 336–344, 2006.
- Q. Su, X. Xu, H. Guo, X. Wu, X. Zhang, B. Swen, and Z. Su. Hidden Sentiment Association in Chinese Web Opinion Mining, *Proceedings of WWW'08*, Beijing, China, pp. 959–968, 2008.
- H. Takamura, T. Inui, and M. Okumura, Extracting semantic orientations of phrases from dictionary, *Proceedings of the Joint Human Language Technology/North American Chapter of the ACL Conference (HLT-NAACL)*, Rochester, NY, 2007.

- L. Tesnière, éléments de syntaxe structurale, Klincksieck, Paris, 1959.
- M. Thomas, B. Pang, and L. Lee, Get out the vote: Determining support or opposition from congressional floor-debate transcripts, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Sydney, Australia, pp. 327–335, 2006.
- I. Titov and R. McDonald, A joint model of text and aspect ratings for sentiment summarization, *Proceedings of 46th Annual Meeting of the Association for Computational Linguistics (ACL'08)*, Columbus, OH, 2008.
- R. M. Tong, An operational system for detecting and tracking opinions in on-line discussion, *Proceedings of the Workshop on Operational Text Classification (OTC)*, New Orleans, LO, 2001.
- P. Turney, Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews, *Proceedings of the Association for Computational Linguistics (ACL)*, Philadelphia, PA, pp. 417–424, 2002.
- X. Wan, Using bilingual knowledge and ensemble techniques for unsupervised Chinese sentiment analysis, *Proceedings of EMNLP08*, Honolulu, HI, pp. 553–561, 2008.
- J. Wiebe, Learning subjective adjectives from corpora, *Proceedings of AAAI*, Austin, TX, 2000.
- J. Wiebe, R. F. Bruce, and T. P. O'Hara, Development and use of a gold standard data set for subjectivity classifications, *Proceedings of the Association for Computational Linguistics (ACL)*, College Park, MD, pp. 246–253, 1999.
- J. Wiebe and R. Mihalcea, Word sense and subjectivity, *Proceedings of the Conference on Computational Linguistics/Association for Computational Linguistics (COLING/ACL)*, Sydney, Australia, pp. 1065–1072, 2006.
- J. Wiebe and T. Wilson, Learning to disambiguate potentially subjective expressions, *Proceedings of the Conference on Natural Language Learning (CoNLL)*, Taipei, Taiwan, pp. 112–118, 2002.
- J. Wiebe, T. Wilson, and C. Cardie, Annotating expressions of opinions and emotions in language, *Language Resources and Evaluation*, 1(2), 165–210, 2005.
- J. Wiebe, T. Wilson, R. Bruce, M. Bell, and M. Martin, Learning subjective language, *Computational Linguistics*, 30, 277–308, 2004.
- T. Wilson, J. Wiebe, and R. Hwa, Just how mad are you? Finding strong and weak opinion clauses, *Proceedings of AAAI*, San Jose, CA, pp. 761–769, 2004.
- T. Wilson, J. Wiebe, and P. Hoffmann, Recognizing contextual polarity in phrase-level sentiment analysis, *Proceedings of the Human Language Technology Conference and the Conference on*

Empirical Methods in Natural Language Processing (HLT/EMNLP), Vancouver, Canada, pp. 347–354, 2005.

H. Yang, L. Si, and J. Callan, Knowledge transfer and opinion detection in the TREC2006 blog track,

Proceedings of TREC, Gaithersburg, MD, 2006.

J. Yi, T. Nasukawa, R. Bunescu, and W. Niblack, Sentiment analyzer: Extracting sentiments about a given topic using natural language processing techniques, *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, Melbourne, FL, 2003.

H. Yu and V. Hatzivassiloglou, Towards answering opinion questions: Separating facts from opinions and identifying the polarity of opinion sentences, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Sapporo, Japan, 2003.

W. Zhang, L. Jia, C. Yu, and W. Meng, Improve the effectiveness of the opinion retrieval and opinion polarity classification. *ACM 17th Conference on Information and Knowledge Management (CIKM)*, Nap Valley, CA, 2008.

W. Zhang and C. Yu, UIC at TREC 2007 Blog Report, 2007.
<http://trec.nist.gov/pubs/trec16/papers/uic-zhang.blog.final.pdf>

Z. Zhang and B. Varadarajan, Utility scoring of product reviews, *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, Arlington, VA, pp. 51–57, 2006.

L. Zhuang, F. Jing, X.-Y. Zhu, and L. Zhang, Movie review mining and summarization, *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, Arlington, VA, 2006.

9. Appendices

A. Prolog

A.1 BEYOND INTRODUCTORY PROLOG

This appendix is not intended to teach you Prolog, but rather to review the language and present some topics that are often skipped in introductory courses. Some knowledge of Prolog is presumed from the beginning.

The emphasis will be on the data—structuring features of Prolog. I won't say much about how these features are used in natural language processing (that's what the rest of the book is for). In fact I will use plenty of examples that do not, by themselves, do anything useful at all. This is intentional. To understand what Prolog actually makes the computer do, it helps to look at the features of the language outside their usual context. An important concern here will be to show that Prolog is simpler than you probably thought. Many details which you probably learned separately will turn out to be instances of just a few systematic principles.

BASIC DATA TYPES

A.2.1 Terms

Prolog is a language for data as much as for procedures. Every data item that can exist

Prolog data items are called TERMS. The syntax of terms in Prolog is on the syntax of expressions in C. Terms are of three kinds: *variables*, *atomic* (NUMBERS and ATOMS), and compound terms (STRUCTURES).

Notice a difference between Prolog and Lisp terminology. In Lisp, the word refers both to symbols such as ABC and to numbers such as 4 5. In Prolog, mom means 'symbol.' Numbers are atomic terms, but they are not atoms.

Atoms normally begin with a lowercase letter and contain letters, digits, and/or character 'p'. Examples are x, xAB cDE, and a 1 anda2 .

Any series of characters enclosed in single quotes is an atom. Thus:

`'this is a long atom with spaces and 0#5!#$ in it'`

A zero—length atom is written

Single quotes occurring within single quotes are written double. In some implemen- tations, backslashes within single quotes are also written double; consult your manual.

An atom consisting entirely of special characters from the set

does not need quotes. Thus : —, — —>, and \ + are (or can be) atoms. (See your manual to confirm that your Prolog uses exactly this set of characters.)

Exercise A.2.1.1

Which of the following are Prolog atoms (symbols)?

abcl23	list_all	list-all	'ab cd'	"ab cd"	°—
— —>	—def—>	23	' 2 3 '	' 2 + 2 '	{2+2}

Exercise A.2.1.2

Which of the items in the previous exercise are atomic terms?

Exercise A.2.1.3

Why are there quotes in wr i t e (' One ') but not in wri L e (one) ?

Exercise A.2.1.4

Are `abc` and `abc` the same atom? Use the computer to make sure of your answer, and state how you did so.

A.2.2 Internal Representation of Atoms

Atoms are not character strings; they are locations in a symbol table. That is, the computer stores only one copy of each atom, no matter how many times it occurs in the program. All occurrences of the atom in the program are then replaced by pointers to its location in the symbol table.

This means that, during program execution, the comparison

```
abracadabraabracazam = abracadabraabrashazam
```

takes no more time than

```
a = b
```

because in each case, the pointers get compared rather than the strings of characters. The real comparison—that is, the determination whether the strings were identical—was done once and for all when the atoms were first placed in the symbol table.

Placing atoms in the symbol table is known as **INTERNiNG** or **+oKENiZING** the atoms. At every moment the symbol table contains all the atoms that exist—whether they are part of the program itself, or data being processed by the program, or anything else.

Exercise A.2.2.1

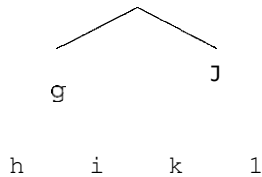
Which of the following built—in Prolog predicates can cause new atoms to be added to the symbol table? Explain why.

```
read/1      write/1      consult/1    reconsult/1  name/2      assert/1
```

A.2.3 Compound Terms (Structures)

A STRUCTuec consists of a *functor* with one or more ARGUMEN+s, such as `f ( a , b , c )`. The arguments can be terms of any type, including other structures. The functor is an atom with a specific number of argument positions; `f` with one argument and `f` with two arguments are not the same functor.

In Prolog, a functor is merely a data—structuring device; it does not refer to a procedure that is to be applied to the arguments. It is just a kind of label. A structure such as $f(g(h, i), (k, l))$ represents a treelike data object that could be diagrammed as:



The ARITY of a functor is the number of arguments it takes. To refer to a functor, instead of saying “f, with 3 arguments” or “f, with arity 3” you can say $f/3$.

When a structure contains other structures, such as $f(a, b(c, d), e)$, the outer—

Draw tree diagrams of the following structures:

`pred(a1,a2,a3,a4)` `g(x(y,z,w),q(p,r(e(f))))` $2+3*4$

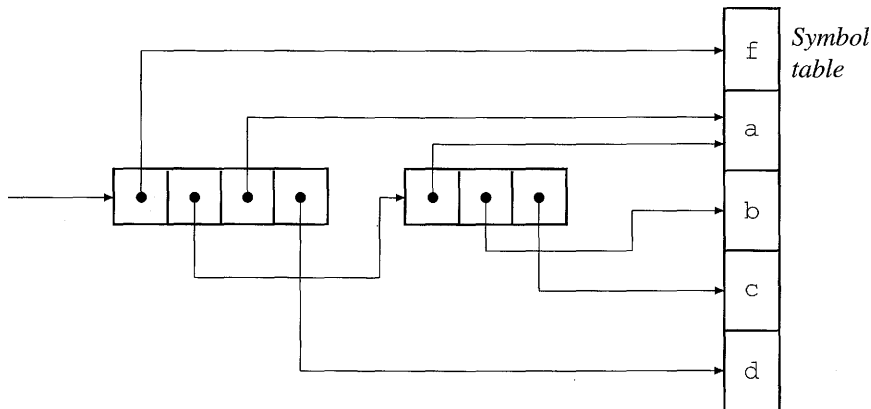
(Hint: Recall that + and * are functors that are written between their arguments.)

Exercise A.2.3.2

Suppose you are creating a Prolog knowledge base about a family tree. Would it sense to use both `parenL / 2` and `parent / 1` in the same program? What would each them mean?

A.2.4 Internal Representation of Structures

In the computer, a structure is a linked tree made of pointers to its substructures and to entries in the symbol table. For example, the structure `f a (b, c), a, d` is represented as:



That is, each structure is represented by a cell containing pointers to the functor and the arguments. The functor is always represented by an atom, i.e., an entry in the symbol table; the arguments can be terms of any type. The cell also contains other information not shown in the diagram, such as the arity of the functor. [If you are not

familiar with this type of diagram, consult a data structures textbook. See also Covington, Nute, and Vellino (1988, ch. 7), and O’Keefe (1990, ch. 3).]

Exercise A.2.4.1

Draw both a tree diagram, and a cell-and-pointer diagram like the diagram above, for each of the following terms:

$f(a, b) + g(c, d)$ $f(f(f(f(f(f(g))))))$ $2 + 3 * 4$

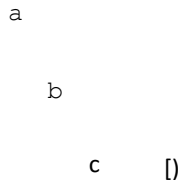
Note that to represent a number, Prolog does not use a pointer; instead it stores the number itself in the cell. Note also that, unlike the symbol table, the tree diagram of $f(f(a))$ contains f in two different places.

Exercise A.2.4.2

Which structure takes more memory, $f(a, b, c, d, e)$ or $f(abcde, abcde, abcde)$? Explain how you can answer this question without knowing the amount of memory needed for a character, a pointer, or a cell on any particular computer.

A.2.5 Lists

A very useful kind of structure takes the form



This structure consists of a , b , and c strung together; it is called a LIST. Here $[]$ (pronounced “nil”) is a special atom that denotes the end of the list. By itself, f denotes an EMPTY LIST (a list with no elements).

Notice that the first argument of each dot functor $(.)$ is an element, and the second argument is another list. That is: *every list consists of an element plus another list or $[]$* . If the whole thing does not end in $[]$ it is called an **imPROPER LIST** (like a dotted pair in Lisp).

There is a shorthand notation for lists, illustrated on the right below:

$.(a, [])$ $[a]$

<code>. (a, . (b, . (c, [])))</code>	<code>[a,b,c]</code>	
<code>. (a,b)</code>	<code>[alb]</code>	<code>% an improper list</code>
<code>. (a, . b, . (c,d))</code>	<code>[a,b,cld]</code>	<code>% another improper list</code>

Every nonempty list has the dot (`.`) as its principal functor.

The big advantage of lists over multiargument structures is that you can process a list without knowing how many elements it has. Any nonempty list will unify with

`. (X, Y)` (more commonly written `[X Y]`), regardless of the number of elements.

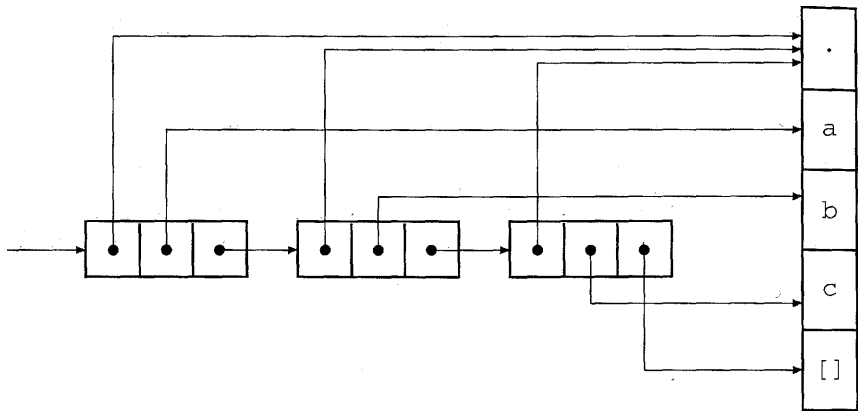
Exercise A.2.5.1

What is the most concise way of writing `!x, y ![]`?

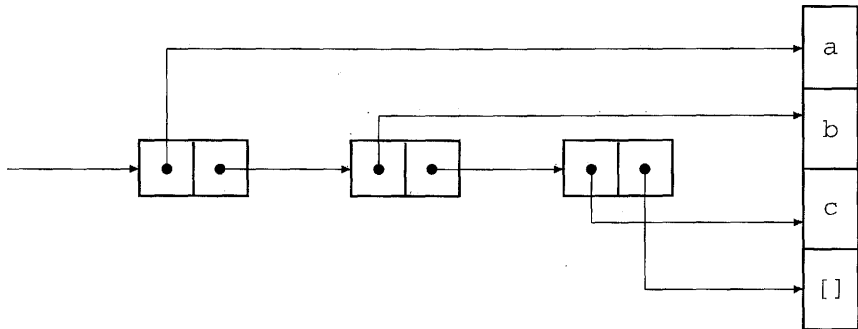
Exercise A.2.5.2

Write `. ( . ( a , . ( b , [ ] ) ) , . ( e , [ ] ) )` in ordinary list notation (with square brackets and

Assuming the dot functor works as just described, the internal representation of the `[a, b, c]` should be:



But lists are very common in Prolog, and all those pointers to the dot functor waste space. Accordingly, almost all Prologs use a more compact representation of lists in which `[a, b, c]` would be represented instead as



which is like the internal representation used in Lisp. That is, the dot functor is, internally, no functor at all.

Regardless of which representation your Prolog uses, lists will behave as if they were held together by the dot functor in the originally described way. But the compact representation saves memory. In particular, it makes `[a, b, c]` take up less space than a listlike structure held together by some functor other than the dot, such as `f(a, f(b, f(c, [])))`

Exercise A.2.6.1

Assuming that each pointer occupies 4 bytes, how much space is saved by representing (a, b, c, d, e, f, g, h, i, j) the compact way instead of the original way?

A.2.7 Strings

A string such as " abc " is simply another way of writing a list of the ASCII codes of the characters, in this case [97 , 98 , 99]. Do not be deceived; a string is a list of numbers, not characters. Like a list, a string can have zero length (written " ").

The built-in predicate `name / 2` interconverts atoms and strings. Some examples:

```
?- name(abc,What) .
What = [97,98,99]           % that is, 'abc'
```

```
?- name(What,'abc') .
What = abc
```

This gives you a way to retrieve the character representation of an atom, or to construct a new atom out of characters strung together. (In C-Prolog and Quintus Prolog, a string that validly represents a number, such as "-234.5", will be converted into a Prolog number, not an atom. In ALS Prolog, ESL Prolog-2, and LPA Prolog, `name / 2` always converts strings into atoms, not numbers.)

Exercise A.2.7.1

What is displayed by the Prolog query `'?- write(•abc).".'?` Why?

Exercise A.2.7.2

In what situation does `name / 2` produce a zero-length atom?

Exercise AT.7N

Show how to use `name / 2` to convert `•abracadabra` into an atom.

Exercise A.2.7.4

Show how to use `name / 2` to find out the **ASCII code** of the letter `q`.

A.2.8 Charlists

The draft ISO standard (Scowen 1992) does not contain **strings**. Instead, programmers are encouraged to use lists of **one-character atoms**, `[l,i,k,e, ,L,h,i,s]`, which I call *charlists*. The obvious advantages of charlists is that they are **always** output in readable form. and they do not rely on ASCII **numeric** codes.

The proposed **standard** calls for a built-in predicate `atom_chars / 2` that interconverts atoms and charlists:

```
?- atom_chars(abc,What) .  
What = [a,b,o]  
  
?- atom_chars(What, (a,b,c)) .  
What = abc
```


what the standard calls for; instead, it interconverts atoms and strings.

Exercise A.2.8.1

Define a recursive procedure `string_chars/2` that will interconvert strings and (its first argument should be a string, and its second argument, a charlist; either can be uninstantiated.) (Hint: only two clauses are needed.)

Exercise A.2.8.2

Using `string_chars/2`, implement `atom_chars/2`. Don't worry about correctness of numbers (e.g., "23").

A.3 SYNTACTIC ISSUES

A.3.1 Operators

Some functors such as `+` can be written between their arguments, in which case no parentheses are needed. Thus `2 + 3` is equivalent to `+(2, 3)`. Such functors are called **INFIX OPERATORS**.

In Prolog, operators need not stand for operations. Thus `2 + 3` does not mean 'add 2 and 3'—it's just a data structure with functor `+` and arguments 2 and 3. The built-in predicate `is`, and some others, can be used to evaluate the structure `2 + 3`, giving the number 5. Apart from this, `2 + 3` has nothing to do with 5. Here Prolog differs from most other programming languages, which evaluate '2+3' as '5' immediately wherever it occurs.

There are also **PREFIX** operators, which come before their arguments, and **POSTFIX** operators, which come after their arguments. An infix operator has two arguments; a prefix or postfix operator has only one.

The built-in predicate `display/1` displays any structure in ordinary (non-infix) notation, treating any operators within it as if they were ordinary functors.

Exercise A.3.1.1

What is displayed by the query `?- display(2 + a, b).`? Try it on the computer and explain your results.

Exercise A.3.1.2

Consult the manual for the Prolog system that you are using, and give examples of predefined prefix and postfix operators (if any).

A.3.2 The Deceptive Hyphen

Unlike the underscore mark, the hyphen (‘—’) cannot occur in an atom or variable name. If you try to put it there, you will get a structure in which the hyphen is an infix operator. That is, `abc —de f` is not an atom; it’s the structure `—(abc, de f)`.

Because structures can occur in most of the places where atoms can occur, the compiler will not notice that you have made a mistake, but you will eventually get unexpected results.

Exercise A.3.2.1

Draw a tree diagram of `Lai s - f unc L or ( a , b )`

Exercise A.3.2.2

Consider the following knowledge base, in which the programmer has put hyphens in `func—tors` even though Prolog does not permit them there.

```
in-usa(X) :- in-texas(X) .           % Caution: risky syntax!
in-usa(X) :- in-georgia(X) .
in-texas(amarillo) .
in-georgia(macon) .
```

In spite of the syntax error, the knowledge base answers queries correctly. Why?

A.3.3 The Dual Role of Parentheses

Parentheses serve two purposes in Prolog: to enclose the arguments of a functor, and to show how a term is to be divided up. The parentheses in `a+(b+c)` show that `b+c` is a term. Otherwise it wouldn't be; the first `+` would join `a` to `b` and the second `+` would join `a+b` to `c`.

You are free to write parentheses around any term at any time. For example, `(( ( a ) ) )` is the same term as `a`. The extra parentheses in

```
?- display( (b,c) ) .
```

show that 'b, c' is to be treated as a single term even though it contains a comma. Otherwise it would be taken as having two arguments.

Exercise A.3.3.1

Demonstrate, using the computer, that $(((a)))$ is equivalent to a .

A.3.4 The Triple Role of Commas

This, of course, begs the question of what that comma is doing if it isn't separating arguments. The comma is, in fact, a right-associative infix operator. The term (a , b , c , d , e , f) is really $a , (b , (c , (d , (e , f))))$ (analogous to $a + (b + (c + (d + (e + f))))$, but with commas instead of pluses). The Prolog rule

$a :- b , c , d .$

¹In early versions of Arity Prolog, $a + (b + c)$ has to be written as $a + (b + c) ,$ because if any functor is written immediately before $($, Arity Prolog assumes that it is a functor of the ordinary kind (not an operator)

$a :- (b, c, d).$

or rather:

a

b

Knowing this structure is crucial if you want to write a Prolog program that processes other Prolog programs.

The comma has yet a third role in lists. In a, b, c the comma is neither an argument separator nor an infix operator; it is a list—element separator. Whenever a Prolog compiler encounters a comma, it has to figure out which of these three roles the comma is playing.

Some nonstandard versions of Prolog use the comma only as the argument separator; they use a as the infix operator, and something else as the list element separator.

Exercise A.3.4.1

Does $a :- b, c, d$ unify with $X :- Y$?

Exercise A.3.4.2

Give an example of a Prolog term in which there are all three kinds of commas.

Exercise A.3.4.3

If a comma is in a position where it can be either an argument separator or an infix operator, which does your computer take it to be? How do you know?

A.3.5 Op Declarations

You can define operators for yourself by specifying the functor you wish to make into an operator, its associativity, and its precedence.

For example, $a + b + c + d$ is interpreted as $((a + b) + c) + d$, so $+$ is said to be **LEFT-ASSOCIATIVE**. And $a + b * c$ is interpreted as $a + (b * c)$, so $+$ is said to have higher **PRECEDENCE** than $*$ (the operator with higher precedence has larger parts of the term as arguments).

To make $+$ into a right-associative infix operator with precedence 500, execute the query:

```
op(500, xfy, ' & ' ) .
```

Here `:` — is what you write at the beginning of a query that is to be *executed*—*not* stored in the knowledge base—as the program is read in. See your Prolog manual for further details.

Note that by itself this `op` declaration does not give any meaning to `'a'`; it merely allows it to be written between its arguments.

Exercise A.3.5.1

Which has higher precedence, `—` or `/` ?

Exercise A.3.5.2

Which has higher precedence, `:` — or the comma?

Exercise A.3.5.3

We said earlier that $(a, b, c) = (a, (b, c))$. Is the comma left-associative or right-associative?

A.4 VARIABLES AND UNIFICATION

A.4.1 Variables

Prolog variable names begin with upper—case letters or `_`. Examples are `X`, `Y`, `Whatever`, `123`.

Like-named variables occurring in the same term are considered to be the same variable. (Recall that a rule or a fact is one term.) Apart from this, variable names have no significance. In fact, in memory, variables have no names, only locations. This explains why, if you ask Prolog to output a term containing variables, the variables will have arbitrary names such as `p0123`.

The anonymous variable, written `'_'`, is considered unique wherever it occurs; successive anonymous variables are not the same variable even if they occur in the same term.

Exercise A.4.1.1

Of the three terms $f(X, Y, Y)$, $f(p3, pq, pq)$, and $f(X, Y, \_)$, which two are equivalent?

A.4.2 Unification

Two terms can be **UNIFIED** (matched) if they are alike or can be made alike by **INSTANTIATING** (giving values to) variables. (Here instantiation includes the ability to make one variable the same as another.) For example:

$f(X, Y)$ unifies with $f(a, b)$ instantiating $X \leftarrow a, Y \leftarrow b$.

$f(X, Y)$ unifies with $f(Z, Z)$ by making X, Y , and Z become the same variable.

$f(X, X)$ does not unify with $f(a, b)$ because Z cannot have two different values

regardless of the order in which the terms are encountered. Much of the power Prolog, particularly for natural language processing, comes from the of unification.

Exercise A.4.2.1

Unify the following pairs of terms, or show why the unification cannot be performed:

$f(X, Y)$ with $f(a, b)$
 $f(X, Y)$ with $g(X, Y)$
 $f(a, b, c)$ with $f(a, X)$
 $f(a, b, c)$ with $f('a', X)$
 $f(a, b, c)$ with $f('a' , X)$
 $f(a, b, c)$ with $f((('a')), X)$
 $(a, b, , d)$ with $Q1234$
 $[a, b, c, d]$ with $X, Y!Z$
 $[a, X, b, Y]$ with $[Z, a, B, b]$
 $[a, X, Y, Y]$ with $Z, a, b, c)$
 $!a, b, c, d$ with $[X, Y, Z]$
 $(2+3)+X$ with $Q+R$
 $(2+3)+X$ with $2+(3+4)$

A.5 PROLOG SEMANTICS

A.5.1 Structure of a Prolog Program

A Prolog program is a file containing clauses and/or queries in the appropriate format for the Prolog reader.

CLAUSES contain information to be stored in the knowledge base. Clauses are of two kinds. FACTs are atoms or structures whose principal functor is not : – or °

–

```

green(kermit) .
ready_to_go .
kermit likes piggy .      (assuming likes is an infix operator)

```

Rules consist of a term, then the symbol `:-`, then one or more terms joined by commas:

```

green(X) :- wet(X) .
piggy likes X :- handsome(X), amphibian(X) .

```

For some purposes the fact `green(kermit)` is equivalent to the rule `green(kermit) :- Ltrue`, where `Ltrue` is a built-in predicate that succeeds with no other action.

The HEAD of a rule is the term to the left of `:-`; the BODY is the term on the right (which can be a series of terms joined by commas). The body of a fact is `Ltrue`.

A PREDICATE is defined by a set of one or more clauses whose heads have the same principal functor. A predicate or one of its clauses is often referred to as a RoCEDURE. Unlike other languages, Prolog lets the same procedure have more than one definition; each clause is one of its definitions. Sometimes the clauses give ways of handling different arguments; sometimes they give alternative ways of handling the same argument.

A QUERv is a request for the computer to do something. When a query occurs in the program file, it is preceded by (with nothing else on the left) thus:

```
- write('Program is loading. ').
```

The Prolog system loads (cONSULTS) a file by reading clauses and queries from it one by one. The clauses get stored in memory; the queries get executed immediately, the moment they are encountered.

The draft ISO standard does not specify how to consult a file. Traditionally, Prolog has had three built-in predicates:

- `c onsul C filename`) reads the file into memory for the first time;
- `r ec onsu1 (filename)` reads the file again after it has been edited; it discards previously existing predicates from memory when it encounters new definitions of the same predicates;
- `c ompi l e filename` , if available, is like `rec onsu1C` except that the code is compiled rather than interpreted, and thus will run faster.

In most Prologs, if you `consul L` the same file twice, you will end up with two copies of each clause in memory. This does not happen with `reconsulC`, but in some older Prologs (such as Arity 4.0), `reconsu1I` requires all the clauses for each predicate to be contiguous, and if they aren't, it will discard all but the last contiguous group. In Quintus Prolog, `c onsul C` and `r ec onsu1 t` are exactly alike; neither of them will produce duplicates if done repeatedly, and neither of them requires clauses to be contiguous (but if style checking is turned on, you will get warnings about discontinuous predicates).

Exercise A.5.1.1

What is the head of each of the following clauses?

```
green(kermit).  
green(X) :- wet(X).  
kermit likes piggy.
```

Exercise A.5.1.2

Explain what happens when the following file is consulted. Assume that the knowledge base is initially empty.

```
- write('Starting'), nl.  
- green(kermit), write('Succeeded the first time'), nl.  
green(kermit).
```

Here is how Prolog executes a query.

- If the query is a series of terms joined by commas, the individual terms are as queries (suBGOALs) and executed one by one, in sequence. That is, to

```
7- green(X), write(X).
```

the Prolog system executes `green(X)` and then `write(X)`. Notice that the query is itself a term, and hence the two `x`'s here are the same variable.

- If the principal functor of the query is a built-in predicate, the system performs the appropriate system—defined action:

```
?- write(hello).
```

- Otherwise, the system finds the first clause whose head can unify with the query and performs the unification. If the clause is a fact, no other action is performed. If the clause is a rule, its right—hand side is treated as a new query. Thus the rule

```
a(X) :- b(X), c(X).
```

transforms the query

```
?- a(zzz).
```

into the query

```
?- b(zzz), c(zzz).
```

A fact by itself can do useful work. The query `?- green(X).` can be answered by the fact `green(kermi)`, which instantiates `X` to `kermi`.

Exercise **A.5.2.1**

Show how the query ? – amphibian (kermit C). ' is transformed into other queries, and ultimately solved, in the following knowledge base:

```
amphibian(X)
```

```
    'Latin name' (X, Genus, Species) ,  
    class ('Amphibia' , Genus) .
```

```
'Latin name' (kermit, 'Rana' , catesbiana) .
```

```
class ('Amphibia' , 'Rana') .
```

Exercise A.5.2.2

Do the same for the query ? – amphib i an (Who) . ’

A.5.3 Backtracking

If at any point there is no clause to match the current query, that query is said to **FAIL**. Execution then backs up to the nearest BACKTRACK POINT (untried alternative), undoing any unifications that took place subsequent to that point, and proceeds forward again along an alternative path.

Alternatives exist because more than one clause can match the same query. When- ever execution enters a clause and there is another clause (as yet untried) that would have matched the same query, the Prolog system records a backtrack point.

Exercise A.5.3.1

Given the knowledge base

$a(X) :- b(X) .$

$a(X) :- c(X) .$

$b(Z) :- d(Z) .$

$b(Z) :- e(Z) .$

$c(Y) :- h(Y) .$

$e(f) .$

show exactly what goals are tried, in what order, to solve the query ? – a (f) . ’ (You may find it convenient to draw a treelike diagram.) Which backtrack points are left untried?

A.5.4 Negation as Failure

If the query $? - p.$ succeeds then the query $? - \backslash + p.$ fails, and vice versa.

The symbol $\backslash +$ is pronounced 'not' and was written not in older Prologs.

A query that fails does not leave a variable instantiated. By definition, either $\backslash + p(X)$ fails or $p(X)$ fails. Therefore the query $? - \backslash + p(X).$ leaves X uninstantiated.

A double $\backslash +$ is a handy way of finding out whether a query would succeed, without instantiating the variables in it. Thus $\backslash + \backslash + p(X).$ succeeds if and only if $p(X)$ would succeed, but does not leave X instantiated.

To force a clause to fail, put $, !$ at the end of it. This is rarely necessary. Usually, rather than write a clause that will fail in a particular situation, you simply refrain from writing a clause that will succeed in that situation.

Exercise A.5.4.1

What do the following clauses do? Suggest a more obvious way of doing the same

```
not_a_list([_l_l] :- !, fail.  
not_a_list(_).
```

A.5.5 Cuts

The `cut` predicate, written `!`, tells the Prolog system to forget about some of backtrack points. Specifically, it discards all backtrack points that have been since execution entered the current clause. This means that, after executing a `cut`,

- it is no longer possible to try other clauses as alternatives to the current clause;
- it is no longer possible to try alternative solutions to subgoals preceding the cut the current clause.

This applies, of course, only to the goal that caused execution to enter the current clause in the first place; it does not permanently change the knowledge base.

An example will make this clearer. Consider the program:

```
/* Clause 1 */  a(X) :- b(X) .  
/* Clause 2 */  a(w) .  
/* Clause 3 */  b(X) :- c(X), !, d(X) .  
/* Clause 4 */  b(y) .  
/* Clause 5 */  b(w) .  
/* Clause 6 */  c(w) .  
/* Clause 7 */  d(z) .
```

The query `b(y)` succeeds. It first matches clause 3, but `c(y)` fails and execution backs out of clause 3 before the cut is executed, and then tries clause 4, successfully.

The query `b(w)` fails. Execution enters clause 3, `c(w)` succeeds, the cut is executed, and then `d(w)` fails. But because of the cut, it is impossible to try any other clauses for `c` or `b`, so the fact `b(w)` is ignored.

The query $a(w) . '$ however succeeds. Clause 1 invokes clause 3, which fails just as before. However, the cut in clause 3 does not impair backtracking out of clause 1, so execution backtracks from clause 1 to clause 2 and then succeeds.

Exercise A.5.5.1

Given the knowledge base

$f(X) :- g(X), !, h(X) .$

$fl(X) :- j(X) .$

$g(a) .$

$j(a) .$

what is the result of executing the query $? - f(a) . '$?

Exercise A.5.5.2

Given the knowledge base:

```
x(X) :- a(X), b(X), '
x(X) ':- c(X).
```

```
a(aa).
```

```
b(aa).
```

```
c(cc).
```

what is the effect of each of the following queries?

```
?- x(aa).
```

```
'- x(cc)
```

```
?- x(What), write(What), nl, fail.
```

Exercise A.5.5.3

Consider the knowledge base:

```
a :- b, my_cut, c(1).
a :- d.
```

```
b.
```

```
c(0).
```

```
d.
```

```
my_cut ':- '
```

What happens when the user types the query `? - a . ' ?` What if `my_cuL` in the first line is changed to `' ?`

A.5.6 Disjunction

If two subgoals are joined by a semicolon (`;`) rather than a comma, they are alternatives. The Prolog system executes the first subgoal and remembers the second subgoal as an untried alternative. Thus

`p :- q ; r.`

is equivalent to

In this book the semicolon is rarely used. It is usually preferable to use alternative

Most Prologs (including the draft ISO standard, but not Arity) have an “if—then” of the form $(p \rightarrow q ; r)$ (pronounced “if p then q else r”). To execute this the computer first tries to execute p. If p succeeds, it then executes q; otherwise executes r. In either case it does not leave a backtrack point.

The “if-then” is not used in this book. Instead of

```
f ← (p → q ; r) .
```

this book uses

```
f :- p, !, q.
f :- r.
```

Arity Prolog has limited-scope cuts called “snips,” written `!!`. Once execution has progressed through the closing snip `!`, it is no longer possible to backtrack to any alternatives within the snips. To implement essentially the same thing in other Prologs, we can define a metapredicate once:²

```
once(Goal) :- Goal,
```

Here Goal can be a compound goal (a set of goals joined by infix commas). So the equivalent of `!, p, q, r, !` is `once( (p, q, r) )`, but `once` can be used in practically any Prolog, not just Arity. In the draft ISO standard, `once` is a built-in predicate.

A.5.8 Self-Modifying Programs

Prolog programs can modify themselves. The built-in predicate `asserta/1` inserts a clause *before* the preexisting clauses for its predicate (if any). For example,

```
?- asserta( (f(X) :- g(X)) ).
```

inserts the rule $f(X) :- g(X)$ in front of the first preexisting clause for $f/1$. (Notice the extra parentheses that are necessary whenever the argument of `asserta` contains `:-`.) The predicate `assertz/1` does the same thing, except that it would put $f(X)$

: - g (X) *after* the preexisting clauses for f / 1.

The built-in predicator `retract / 1` removes a clause from the knowledge base. Specifically, it removes the first clause that matches its argument, or fails if there is no such clause. For example,

```
?- retract (green (X)) .
```

²Some older Prologs require Goal on the right-hand side to be written as `call (Goal)` .

removes the first clause that matches `green(X)`, simultaneously unifying `X` with what—ever was in the argument position. Because of this unification, `retract` can be used to retrieve information while deleting it from the knowledge base. Like `asserta`, `assertz`, and all other Prolog functors, `retract` requires extra parentheses if its argument contains `' : - '` or a comma.

To retract all the clauses of `f/2`, you can use

```
?- abolish(f/2).
```

(or in ALS Prolog, `abolish(1,2)`).

One extremely important point must not be forgotten:

`asserta`, `assertz`, and `retract` are not for use in a way of doing temporary data

Their use is justified only in situations that reflect a genuine, permanent change to the knowledge contained in the program. The normal way to hold information temporarily (analogous to storing it in a Pascal or C variable) is to pass it along in the arguments of procedures (see the section on repetitive algorithms below).

Exercise AT.8.1

What would be in the knowledge base after execution of the following four queries? Assume that beforehand, there were no clauses for `green`.

```
?- asserta(green(kermit)).
```

```
?- asserta(green(eggs+ham)).
```

```
?- assertz(green(cheese)).
```

```
?- asserta(green(grass)).
```

Exercise A.5.8.2

Which of the following queries will successfully retract the clause ' $f(X) :- g(X), h(X).$ '? Explain why.

```
?- retract(f(X)).  
?- retract(tf(W) :- g(W), hTV)).  
?- retract((f(WXYZ) :- g(WXYZ), h(Q))).  
?- retract(f(X) :- Y).  
?- retract((f(X) :- Y)).  
?- abolish(f/1).  
?- abolish(f/3).
```


In Quintus Prolog and in the draft ISO standard, if any of the clauses for a predicate to be asserted or retracted, the program must contain a declaration such as

– `dynamic f/2.`

(where `f/2` stands for predicate `f` taking 2 arguments) before the clauses for the (if any). This tells the compiler not to compile the clauses into a form that cannot be recognized at run time. In fact Quintus does not compile them at all; it runs interpretively.

The only exception is that if a predicate is created entirely by asserting, it is automatically dynamic and need not be declared.

Dynamic declarations are not required by Prolog; you can abolish any predicate, even a compiled one.

A.6 INPUT AND OUTPUT

The input-output system described here is the traditional one from Edinburgh (DEC-IO) Prolog. Virtually all present-day Prolog implementations support it, but the draft ISO standard introduces a different input-output system only partly compatible with the original one. When in doubt, check your manual.

A.6.1 The Prolog Reader

Built into Prolog is a procedure for reading terms from input devices (files, the keyboard, etc.) and converting them into their internal representations. This procedure is called the `READER` and you can access it through the built-in predicate `read/1`. The reader is also used by `consult` and `reconsult` and by the routine that accepts queries that you type on the keyboard.

`read/1` reads, from the standard input file, exactly one Prolog term followed by a period, then unifies its argument with this term. The term can be of any kind whatsoever. The reason for ending with a period is that a term can occupy more than one line.

After the period there must be a blank, or a comment, or the end of the line; thus the period in the middle of the term `3.2` does not denote end of term.

The character `%` (not in quotes) makes the reader ignore everything until the

begin- ning of the next line. The sequence / * makes it skip everything until after the following

* /. Thus both % and / * * / provide ways to delimit cOMMENTS. Comments are permitted not only in programs, but in everything that the reader reads. Generally, however, you cannot put a comment within another comment.

Extra blanks and comments are permitted anywhere, so long as they do not interrupt a number, atom, functor, or quoted string, nor come between a functor and the parenthesis that introduces its arguments. Thus $f(a)$ can be written $f(a)$ but not as $I(a)$.

The reader recognizes variables but does not preserve their names. It *does* notice when two like-named variables occur in the same term. Thus, if you type `[x, X]` you get a list with two instances of the same variable in it. Notice that a Prolog clause such as

```
f(X) :- g(X), h(X).
```

is all one term, which is why all the occurrences of `X` in it are treated as the same variable.

When the reader tries to read past the end of a file, it behaves as if it has read the atom `end_of_file`. This is of course also what it reads if it encounters the characters

```
end_of_file.
```

which you can insert in the middle of a file to make the reader stop reading there.

Exercise A.6.1.1

What does `read('yes').` do if the user types `'yes.'`? What if the user types `'no.'` or `'Yes.'`?

Exercise A.6.1.2

In your Prolog, is `a/*cornaenL*/b` treated like `ab` or like `a b`? State how you tested this.

Exercise A.6.1.3

What is printed by the query `?- Sri L e (f(X, Y, X)).`? Why? What does the reader have to do with this?

Exercise A.6.1.4

In the Prolog clauses

```
f(x) :- h(X).
```

$g(X) :- z(X).$

which occurrences of X are the same variable? Explain why the reader is responsible for this.

A.6.2 The Writer

Corresponding to the reader there is, of course, a `wRITER` Whose job is to produce a written representation of any term. The writer is usually accessed through `Sri Ce / 1`. Note, however, a possible problem:

```
?- write('hello there').  
hello there
```

In this case the term `'be ll o Cher e'` is written out as `Set l o Tier e`, which is not a term. Thus, in this case, output written by `wri Le` is not suitable to be read back in

The built-in predicate `writeln` remedies this problem. Everything that it writes in legal Prolog syntax:

```
?- writeln('hello there').
'hello there'
```

```
?- writeln(2+3-4).
2+3-4
```

But there is still a potential problem. Suppose you have defined `likes` to be infix operator. Then you can do the following:

```
?- writeln(kermit likes piggy).
kermit likes piggy
```

The reader will accept `kermit likes piggy` if it knows that `likes` is an infix operator. A subsequent program, reading this term back in from a file, would encounter difficulties if it did not have the same infix operator defined.

A third built-in predicate, `display`, remedies this problem too: `display` writes terms so that they can be read back in without relying on any operator declarations at all. For example:

```
?- display(2+3-4).
'- '(+'(2,3),4)
```

```
?- display(kermit likes piggy).
likes(kermit,piggy)
```

Finally, remember that if you are writing out terms to a file so that the reader can subsequently read them, you must put a period after each term.

A.6.3 Character Input-Output

The built-in predicate `putchar` outputs the character whose ASCII code is given as its argument. For example, `?- putchar(65).` outputs the letter A.

The built-in predicate `getchar` accepts a character from the input stream and unifies its argument with the ASCII code of that character. For example, `?- get0(X).` reads one character and unifies its code with `x`, and `?- get0(65).` reads a character and succeeds if that character is 'A' or fails otherwise.

Another built-in predicate, `get f 1`, works like `get 0` except that it Skips over any nonprinting characters (blanks, returns, line feeds, etc.) and reads the first nonblank character.

Exercise A.6d.1

Define a predicate called `copy 3` that accepts three characters from the keyboard and writes their ASCII *codes* (is numbers) on the screen. Experiment with its behavior using both `get` and `get 0`.

A.6.4 File Input-Output

Prolog normally reads from the keyboard and writes to the screen. These are referred to as STANDARD INPUT and STANDARD OUTPUT, respectively. Standard input and output can be REDIRECTED to files or to anything else that the operating system can treat as a file (such as a printer).

To redirect standard input, use `see ( filename )`; to redirect standard output, use `tell ( filename )`, where `filename` is a Prolog atom giving the name of the file. To close the files and cancel the redirection, use `seen` and `tell`, respectively.

Whenever input or output is redirected to a file, the file is opened for reading or writing as the case may be. This means that redirected input always starts at the beginning of the file, and redirected output always starts by creating the file (and destroying any like-named file that already exists). It is, however, possible to have many files open at once. Here is an example:

```
test .

see(file1),          % start at beginning of file1
read(T1),            % read a term

see(file2),          % start at beginning of file2
read(T2),            % read a term

see(file1),          % switch back to file1
read(T3),            % read second term on file1
write([T1,T2,T3]),   % write results on the screen
seen.                % close all files
```

Use `see (user)` or `tell (user)` to redirect input or output to the keyboard or screen without closing files. Use `seeing (F)` or `telling (F)` to find out the name of the file that input or output is currently using. Thus, the following code writes a message

on the screen, then switches subsequent output back to wherever it had been redirected:

```
telling(F),  
tell(user),  
write('This is a message'),  
tell(F),
```

Redirection can be dangerous, because if the program crashes while input or output is redirected, you may be unable to see the error messages or type any further input to the Prolog system. Almost all Prologs provide a way to read and write files without redirecting standard input or output. Unfortunately, such methods are not standardized and therefore cannot be used in this book.

All the built-in input-output predicates are DERMINISTIC; that is, they do not yield

Under what conditions must a file name be given in quotes?

Exercise A.6.4.2

Write a short program that accepts three terms (each followed by a period) from the keyboard, writes them out to a file named ECHO.DAT, and then reads them back in and displays them on the screen.

Exercise A.6.4.3

What does `?- read(X), fail, !, !.` do?

1.7 EXPRESSING REPETITIVE ALGORITHMS

A.7.1 repeat loops

The built-in predicator `repeat` is considered to have an infinite number of different solutions, so that execution can backtrack to it and go forward again an infinite number of times:

```
?- repeat,  
    write('I run forever...'),  
    fail.
```

More commonly, of course, the repetition is stopped by executing a cut after some test is passed. Here is a predicate that copies terms from standard input to standard output, up to and including end of file:

```
copy_terms :-  
    repeat,  
        read(X),  
        write(X), nl,  
        X == end_of_file, % the test
```

Without the cut, repetition would still stop when $X == \text{endpoffile}$ succeeds (because Prolog only backtracks from failures, not successes), but a subsequent failure might cause repetition to resume, as in a query like `copyL erms , z . ' if z fails`.

repeat loops are not as useful as you might think, because there is no way to carry information along from one pass to the next. All variable instantiations are undone as you backtrack through them. Also, execution will backtrack from any query that fails—not just the one you think of as the test.

In fact, the main use of repeat loops is to sift through input. There is a good reason for this. Something external has to change, from one repetition to the next, in

order to ensure that the looping will terminate. So an repeat loop that does not look at something external, such as a file, will rarely be useful.

Exercise A.7.1.1

The following predicate is supposed to copy characters from standard input to standard output until a blank is encountered. After copying the blank it should stop.

```
copy_until_blank - % Contains error!
repeat,
    get0(Char),
    put(Char),
[Char] -- " "
```

What did the programmer do wrong? Show how to demonstrate that the program does not work entirely as desired.

Exercise A.7.1.2

Define a predicate `get_lower_case_letter/1` that is like `get` except that, instead of just skipping nonprinting characters, it skips everything that is not a lower-case letter.

For example, `?-get_lower_case_letter(X)` should find the first lower-case letter on the input stream and instantiate `X` to the ASCII code of that letter. Likewise,

`?-get_lower_case_letter(100)` should read the first lower-case letter on the input stream and succeed if that letter was code 100 ('d') or fail otherwise.

A.7.2 Recursion

To do something repeatedly and carry information along from one pass to the next, use **RECURSION**—that is, make a procedure call itself. This is just like induction in logic. (Formal logic does not allow you to just “repeat” a line in a proof; instead, you must define one

thing in terms of another so that you get the repetition you need.) Here's how to print the integers from 1 to 100:

```
count from(X)
  X > 100.                % succeed with no further action

count from(X) -
  X =< 100,                % print this number, compute next one
  write(X), nl,
  NewX is X+1,
  count from(NewX) .

?- count_from(1) .
```

Here X is called a s ATE VARIABLE because it passes along information about the state of the computation. (State variables are also called AcCouULATORS because they accumulate information.) Note that there's no way to change the value of X once it's instantiated,

to define a second predicate (traditionally named something-aux and called an ILIARY PREDICATE). The first predicate has only the arguments that are meaningful the user, and it calls the aux predicate, which has the full set of arguments. example:

```
reverse(List1,List2) :-reverse_aux(List1,[],List2).

reverse_aux([H|T],Stack,Result)
    reverse_aux(T,[H]Stack,Result).

reverse_aux([],Result,Result).
```

This is a classic list-reversal algorithm.

Exercise A.7.2.1

In reverse aux/3 above, how does Result come to be instantiated?

Exercise A.7.2.2

Define a predicate add_up(X,Y,Z) which unifies Z with the sum of the integers from X to Y inclusive. Do this by actually adding the integers, not by using a formula.

A.7.3 Traversing a List

Here's an example of how to work through a list ("CDR down it," in Lisp parlance). The task, in this case, is to make a copy of the list in which every occurrence of a is replaced with b. The program logic will be very similar no matter what you are looking for in the list, and whether or not you are making an altered copy.

```
rewrite(a,b) :- '      , a rewrites as b
rewrite(X,X).      % anything else rewrites as itself

rewrite_list([],[]).

rewrite_list([First|Rest],[NewFirst|NewRest])
    rewrite(First,NewFirst),
    rewrite_list(Rest,NewRest).
```

An example of its use:

```
?- rewrite_list([a,b,r,a,c,a,d,a,b,r,a],What).  
What = [b,b,r,b,c,b,d,b,b,r,b]
```

The logic here is: To do something to a whole list, do it to the first element, then do it to the list of remaining elements.

Exercise A.7.3.1

Define a recursive predicate `maximum (L i s t , N)` which unifies `N` with the largest integer contained in `L i s t`. Assume that all the elements of `E i s t` are integers.

Exercise A.7.3.2

Define a recursive predicate `depth ( L i s t , N)` which unifies `N` with the depth of a list, defined as follows:

- Any term that is not a list has depth 0.
- The depth of any list is 1 + the depth of its deepest element.

Thus the depth of `a` is 0; the depth of `[ a , b ]` is 1; the depth of `[ a , [ b , c ] ]` and of `f a , b ] , c` is 2; and so on.

A.7.4 Traversing a Structure

Often, instead of just lists, you'll need to process arbitrary terms, examining and possibly changing everything that is inside them. For example, if you implement an extension to Prolog (as was done in Chapter 5), you'll want to read every term in a program, searching for your special symbols and converting them into something else, while leaving the rest alone.

Here's how to generalize `ear i L ep1 i s t / 2` so it works for any term:

```
% rewrite_term(+Term1,?Term2)

% Copies Term1 changing every atom or functor 'a' to 'b'

% (using rewrite/2 from previous section).

rewrite_term(X,X)
var(X),
```

```

rewrite term(X,Y)
    atomic(X),          % don't alter uninstantiated variables

    rewrite(X,Y).

                                % 'atomic' means atom or number

rewrite term(X,Y)

    X =.. XList,          % convert structures to lists
    rewrite aux(XList,YList), % process them

    Y =.. YList.          % convert back to structures

rewrite aux([], []).

rewrite_aux((FirstlRest], (NewFirstlNewRest])
    rewrite_term(First,NewFirst), % note recursion here

```


f, a, b, c . Either of the arguments of $=$ can be instantiated, and the other one be given the appropriate value.

Even lists can be converted this way. For example,

```
[a,b,c] =.. ['.',a,[b,c]]
```

because the principal functor of every list is the dot. Thus, lists are a special case of structures and do not require a clause of their own in `rewriter/2`.

Notice that `rewriter` (used by `rewriter`) is like `rewriter`, which we defined earlier, except that it uses `rewriter` recursively on the first element of each list. This ensures that structures within structures are all processed.

Exercise A.7.4.1

Define `depth(S, N)` such that N will be unified with the depth of `S`. This time depth is defined as follows:

- An atomic term has depth 0.
- The depth of a structure is 1 + the depth of its deepest argument.

For example, the depth of `f(a, b(c, d(e, f)), g)` is 3.

Exercise A.7.4.2

What is the depth of a ten-element list according to the definition just given?

A.7.5 Arrays in Prolog

Prolog has no arrays. However, it has two reasonable substitutes. First, if an array is to be used as a lookup table, replace it with a set of facts. For example:

```
nchprime(1,2).
nthprime(2,3).
nthprime(3,5).
nthprime(4,7).
```

This is a table for looking up the nth prime number. You can build it by executing a program that finds primes and uses as s ert z.

Second, a multi-argument structure acts like an array of Prolog terms or variables. For example,

```
f(A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y)
```

is like a 25—element array of variables; you can pass it around and instantiate or test the variables one by one.

To get to the *n*th argument without unifying the whole structure with anything, use the built—in predicate `arg(N, St ruc for e, Arg)`, which unifies `Arg` with the *N*th argument of `St ruc Our e`. For example,

```
°- arg(15, f(a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y), W) .
```

unifies `N` with `o`. This is much faster than putting the whole structure through the unification process.

Exercise A.7.5.1

In C, a two-dimensional array is simply a one-dimensional array of one-dimensional arrays. What does this suggest about how to represent a two-dimensional array in Prolog? Give an example of a 3 x 3 matrix implemented this way.

Exercise A.7.5.2

Prolog implementations put a limit on the number of arguments in a structure—typically

256. What is the limit in the Prolog that you use? How did you find out?

A.8 EFFICIENCY ISSUES

A.8.1 Tail Recursion

Recursion eats up memory because, whenever one procedure invokes another, a record has to be kept of where to come back to when the second procedure finishes.

There is an exception: suppose the first procedure calls the second as its very *last step*, with no further actions to be performed and no backtrack points to be remembered. Then execution can simply jump into the second procedure without setting up a way for control to return to the first one. (At the end, control will instead

end up wherever it was to have gone at the end of the first procedure.)

Recursion that satisfies this condition is called **TAIL RECURSION**. An example:

```
f(X) :- X < 100.                                % Tail recursive example
f(X) :- X >= 100, Y is X-1, f(Y).
```

If the recursive call isn't last, the procedure isn't tail recursive:

```
g(X) :- X >= 100, Y is X-1, g(Y), z.            % Not tail recursive
```

Nor is it tail recursive if there's still an untried alternative at the time the recursive call takes place. This is the case if the clauses in the example are in the opposite order:

```
h(X) :- X >= 100, Y is X-1, h(Y).              % Not tail recursive
```

When the first rule calls itself, the second rule hasn't yet been tried, so a backtraek has to be stored on the stack. As the first rule calls itself over and over, the sack eventually fill up. Try a query like `?- h ( 10 0 0 0 ) .` ' to see what happens.

In almost all Prologs, a cut will make such a procedure tail recursive after ali, eliminating the untried alternative:'

```
j(X) :- K >= 100, !, Y is K-1, j(Y).      % Tail recursive
j(X) : ] < 100.                          % because of cut
```

Place the cut so that it will be executed as soon as it is certain that execution has entered the right clause.

Exercise A.8.1.1

Rearrange the clauses in the following predicate definition and add cuts so that the predicate is tail recursive and never leaves behind an unnecessary backtrack point.

```
% contains_list(+L)

% succeeds if L is a list that contains another list

contains_list([f]l_]).

contains_list([_!X]) :- contains_list(X).

contains_list([[_]_l_]).
```

A.8.2 Indexing

Almost all Prologs can eliminate unnecessary backtrack points by looking ahead to see which clauses the query can match. For example, given the knowledge base

```
f(a(x),Y) :- f(X,Y).
f(b(x),Y) :- g(X,Y).
```

and the query '? - f (a (something) , something) . ' the Prolog system will look ahead and see that the second clause cannot possibly match this query, so when it enters the first clause, it will not leave behind a backtrack point. As a result, this predicate becomes tail recursive even though it does not appear to be.

This feature is called *indexing*. Obviously, the Prolog system cannot predict with complete certainty which clauses will succeed; all it can do is rule out some obvious mismatches in advance.

Indexing normally looks only at the principal functor of the first argument. Thus indexing will distinguish $f(a(x), Y)$ from $I(b(x), Y)$, but it will not distinguish

³Exceptions include Arity Prolog Interpreter 4.0 and ESL Public Domain Prolog—2 version 2.3.5. In these implementations, the cut keeps the backtrack point from being used, but doesn't prevent it from being

$f(Y, a(X))$ from $f(Y, b(X))$. This means that in order to take advantage of indexing you should design your predicates so that the first argument has a variety of principal functors and is usually instantiated when the predicate is called.

Exercise A.8.2.1

Is the predicate

```
print_list([HIT]) :- write(H), put(32), print_list(T).
print_list([]).
```

tail recursive in a system with indexing? Why or why not? (And what does `put(32)` accomplish?)

Exercise A.8.2.2

By changing only the order of arguments, make the predicate

```
p(Y, [Y]).
```

tail recursive, assuming first-argument indexing.

A.8.3 Computing by Unification Alone

Sometimes unification, by itself, can do the computation you need. Here's a predicate that tests whether its argument is a three—element list, and succeeds if so:

```
has_three_elements([_,_,_]).
```

Note that there is no call to `length` or anything like that. Unification does all the work.

Likewise, here's a predicate that takes any list and creates another list just like it with the first two elements swapped:

```
swap_first_two([A,B|Rest],[B,A|Rest]).
```

For example, the query `?- swap_first_two(X,Y,Z,that).` will instantiate `that` to `[y,x,z]`

Exercise A.8.3.1

Define a predicate which, using unification alone, will convert $2+3$ to $2-3$, or $(3+4)+f(x)$ to $(3+4)-f(x)$, or, generally, any structure with principal functor `+` to the corresponding

Programs run more efficiently if they can avoid creating new structures and lists activity known to Lisp programmers as “consing”).

It is much faster to instantiate a previously uninstantiated part of a structure, to make a new copy of the whole structure in order to add information.

For example, if you want to build the list `(a b c d)`, it is much faster to write `(a b c X)` and instantiate `X` to `(d)`, rather than starting with `(a b o)` and appending `(d)` to it.

A list with an uninstantiated tail, such as `(a b c !X)`, is called an OPEN LIST. Often you'll want to keep, outside the list, a pointer to the uninstantiated tail, so that you can get to the tail without searching through all the elements. This is called a DIFFERENCE LIST; an example is `(a b c !X)`, `X`. Here `I` has no particular meaning; you could use any functor. Many people use the infix operator `-` or `/`.

The traditional predicate `append` / 3 can be a time waster; whenever you find yourself using it, see if the same work can be done more quickly with open lists or difference lists.

Another time waster is `univ` (“univ”), which builds a list from a structure. If you only need to retrieve part of the structure, it is much quicker to use `arg` (`N`, `Structure`, `Arg`).