

Allen 1995: Natural Language Understanding

	<u>Contents</u>	<u>Preface</u>	<u>Introduction</u>	
<u>previous chapter</u>	<u>Part I Syntactic Processing</u>	<u>Part II Semantic Interpretation</u>	<u>Part III - Context / World Knowledge</u>	<u>next chapter</u>
	<u>Appendices</u>	<u>Bibliography</u>		
	<u>Summaries</u>	<u>Further Readings</u>	<u>Exercises</u>	

1. Introduction to Natural Language Understanding

	<u>1.1 The Study of Language</u>
	<u>1.2 Applications of Natural Language Understanding</u>
	<u>1.3 Evaluating Language Understanding Systems</u>
	<u>1.4 The Different Levels of Language Analysis</u>
	<u>1.5 Representations and Understanding</u>
	<u>1.6 The Organization of Natural Language Understanding Systems</u>
	<u>Summary</u>
	<u>Related Work and Further Readings</u>

[1.1 The Study of Language](#)

[Exercises for Chapter 1](#)

[Allen 1995 : Chapter 1 - Introduction / 1]

This chapter describes the field of natural language understanding and introduces some basic distinctions. Section 1.1 discusses how natural language understanding research fits into the study of language in general. Section 1.2 discusses some applications of natural language understanding systems and considers what it means for a system to understand language. Section 1.3 describes how you might evaluate whether a system understands language. Section 1.4 introduces a few basic distinctions that are made when studying language, and Section 1.5 discusses how computational systems often realize these distinctions. Finally, Section 1.6 discusses how natural language systems are generally organized, and introduces the particular organization assumed throughout this book.

1.1 The Study of Language

Language is one of the fundamental aspects of human behavior and is a crucial component of our lives. In written form it serves as a long-term record of knowledge from one generation to the next. In spoken form it serves as our primary means of coordinating our day-to-day behavior with others. This book describes research about how language comprehension and production work. The goal of this research is to create computational models of language in enough detail that you could write computer programs to perform various tasks involving natural language. The ultimate goal is to be able to specify models that approach human performance in the linguistic tasks of reading, writing, hearing, and speaking. This book, however, is not concerned with problems related to the specific medium used, whether handwriting, keyboard input, or speech. Rather, it is concerned with the processes of comprehending and using language once the words are recognized. Computational models are useful both for scientific purposes — for exploring the nature of linguistic communication — and for practical purposes — for enabling effective human-machine communication.

Language is studied in several different academic disciplines. Each discipline defines its own set of problems and has its own methods for addressing them. The linguist, for instance, studies the structure of language itself, considering questions such as why certain combinations of words form sentences but others do not, and why a sentence can have some meanings but not others. The psycholinguist, on the other hand, studies the processes of human language production and comprehension, considering questions such as how people identify the appropriate structure of a sentence and when they decide on the appropriate meaning for words. The philosopher considers how words can mean anything at all and how they identify objects in the world. Philosophers also

consider what it means to have beliefs, goals, and intentions, and how these cognitive capabilities relate to language. The goal of the computational linguist is to develop a computational theory of language, using the notions of algorithms and data structures from computer science. Of course, to build a computational model, you must take advantage of what is known from all the other disciplines. Figure 1.1 summarizes these different approaches to studying language.

[Allen 1995 : Chapter 1 - Introduction / 2]

Discipline	Typical Problems	Tools
Linguists	How do words form phrases and sentences? What constrains the possible meanings for a sentence?	Intuitions about well-formedness and meaning; mathematical models of structure (for example, formal language theory, model theoretic semantics)

Discipline	Typical Problems	Tools
Psycholinguists	How do people identify the structure of sentences? How are word meanings identified? When does understanding take place?	Experimental techniques based on measuring human performance; statistical analysis of observations
Philosophers	What is meaning, and how do words and sentences acquire it? How do words identify objects in the world?	Natural language argumentation using intuition about counter-examples; mathematical models (for example, logic and model theory)
Computational Linguists	How is the structure of sentences identified? How can knowledge and reasoning be modeled? How can language be used to accomplish specific tasks?	Algorithms, data structures; formal models of representation and reasoning; AI techniques (search and representation methods)

Figure 1.1 The major disciplines studying language

As previously mentioned, there are two motivations for developing computational models. The scientific motivation is to obtain a better understanding of how language works. It recognizes that any one of the other traditional disciplines does not have the tools to completely address the problem of how language comprehension and production work. Even if you combine all the approaches, a comprehensive theory would be too complex to be studied using traditional methods. But we may be able to realize such complex theories as computer programs and then test them by observing how well they perform. By seeing where they fail, we can incrementally improve them. Computational models may provide very specific predictions about human behavior that can then be explored by the psycholinguist. By continuing in this process, we may eventually acquire a deep understanding of how human language processing occurs. To realize such a dream will take the combined efforts of linguists, psycholinguists, philosophers, and computer scientists. This common goal has motivated a new area of interdisciplinary research often called cognitive science.

The practical, or technological, motivation is that natural language processing capabilities would revolutionize the way computers are used. Since most of human knowledge is recorded in linguistic form, computers that could understand natural language could access all this information. In addition, natural language interfaces to computers would allow complex systems to be accessible to

[Allen 1995 : Chapter 1 - Introduction / 3]

BOX 1.1 Boxes and Optional Sections

This book uses several techniques to allow you to identify what material is central and what is optional. In addition, optional material is sometimes classified as advanced, indicating that you may need additional background not covered in this book to fully appreciate the text. Boxes, like this one, always contain optional material, either providing more detail on a particular approach discussed in the main, text or discussing additional issues that are related to the text. Sections and subsections may be marked as optional by means of an open dot (o) before the heading. Optional sections provide more breadth and depth to chapters, but are not necessary for understanding material in later chapters. Depending on your interests and focus, you can choose among the optional sections to fill out the core material presented in the regular sections. In addition, there are dependencies between the chapters, so that entire chapters can be skipped if the material does not address your interests. The chapter dependencies are not marked explicitly in the text, but a chart of dependencies is given in the preface.

everyone. Such systems would be considerably more flexible and intelligent than is possible with current computer technology. For technological purposes it does not matter if the model used reflects the way humans process language. It only matters that it works.

This book takes a middle ground between the scientific and technological goals. On the one hand, this reflects a belief that natural language is so complex that an ad hoc approach without a well-specified underlying theory will not be successful. Thus the technological goal cannot be realized without using sophisticated underlying theories on the level of those being developed by linguists, psycholinguists, and philosophers. On the other hand, the present state of knowledge about natural language processing is so preliminary that attempting to build a cognitively correct model is not feasible. Rather, we are still attempting to construct any model that appears to work.

The goal of this book is to describe work that aims to produce linguistically motivated computational models of language understanding and production that can be shown to perform well in specific example domains. While the book focuses on computational aspects of language processing, considerable space is spent introducing the relevant background knowledge from the other disciplines that motivates and justifies the computational approaches taken. It assumes only a basic knowledge of programming, although the student with some background in linguistics, artificial intelligence (AI), and logic will appreciate additional subtleties in the development.

>> [back](#)

1.2 Applications of Natural Language Understanding

A good way to define natural language research is to consider the different applications that researchers work on. As you consider these examples. It will

[Allen 1995 : Chapter 1 - Introduction / 4]

also be a good opportunity to consider what it would mean to say that a computer system understands natural language. The applications can be divided into two major classes: text-based applications and dialogue-based applications.

Text-based applications involve the processing of written text, such as books, newspapers, reports, manuals, e-mail messages, and so on. These are all reading-based tasks. Text-based natural language research is ongoing in applications such as

- finding appropriate documents on certain topics from a database of texts (for example, finding relevant books in a library)
- extracting information from messages or articles on certain topics (for example, building a database of all stock transactions described in the news on a given day)
- translating documents from one language to another (for example, producing automobile repair manuals in many different languages)
- summarizing texts for certain purposes (for example, producing a 3-page summary of a 1000-page government report)

Not all systems that perform such tasks must be using natural language understanding techniques in the way we mean in this book. For example, consider the task of finding newspaper articles on a certain topic in a large database. Many, techniques have been developed that classify documents by the presence of certain keywords in the text. You can then retrieve articles on a certain topic by looking for articles that contain the keywords associated with that topic. Articles on law, for instance, might contain the words "*lawyer*", "*court*", "*sue*", "*affidavit*", and so on, while articles on stock transactions might contain words such as "*stocks*", "*takeover*", "*leveraged buyout*", "*options*", and so on. Such a system could retrieve articles on any topic that has been predefined by a set of keywords. Clearly, we would not say that this system is understanding the text; rather, it is using a simple matching technique. While such techniques may produce useful applications, they are inherently limited. It is very unlikely, for example, that they could be extended to handle complex retrieval tasks that are easily expressed in natural language, such as the query "*Find me all articles on leveraged buyouts involving more than 100 million dollars that were attempted but failed during 1986 and 1990*". To handle such queries, the system would have to be able to extract enough information from each article in the database to determine

whether the article meets the criteria defined by the query; that is, it would have to build a representation of the information in the articles and then use the representation to do the retrievals. This identifies a crucial characteristic of an understanding system: it must compute some representation of the information that can be used for later inference.

Consider another example. Some machine translation systems have been built that are based on pattern matching; that is, a sequence of words in one language is associated with a sequence of words in another language. The

[Allen 1995 : Chapter 1 - Introduction / 5]

translation is accomplished by finding the best set of patterns that match the input and producing the associated output in the other language. This technique can produce reasonable results in some cases but sometimes produces completely wrong translations because of its inability to use an understanding of content to disambiguate word senses and sentence meanings appropriately. In contrast, other machine translation systems operate by producing a representation of the meaning of each sentence in one language, and then producing a sentence in the other language that realizes the same meaning. This latter approach, because it involves the computation of a representation of meaning, is using natural language understanding techniques.

One very attractive domain for text-based research is story understanding. In this task the system processes a story and then must answer questions about it. This is similar to the type of reading comprehension tests used in schools

and provides a very rich method for evaluating the depth of understanding the system is able to achieve.

Dialogue-based applications involve human-machine communication. Most naturally this involves spoken language, but it also includes interaction using keyboards. Typical potential applications include

- question-answering systems, where natural language is used to query a database (for example, a query system to a personnel database)
- automated customer service over the telephone (for example, to perform banking transactions or order items from a catalogue)
- tutoring systems, where the machine interacts with a student (for example, an automated mathematics tutoring system)
- spoken language control of a machine (for example, voice control of a VCR or computer)
- general cooperative problem-solving systems (for example, a system that helps a person plan and schedule freight shipments)

Some of the problems faced by dialogue systems are quite different than in text-based systems. First, the language used is very different, and the system needs to participate actively in order to maintain a natural, smooth-flowing dialogue. Dialogue requires the use of acknowledgments to verify that things are understood, and an ability to

both recognize and generate clarification sub-dialogues when something is not clearly understood. Even with these differences, however, the basic processing techniques are fundamentally the same.

It is important to distinguish the problems of speech recognition from the problems of language understanding. A speech recognition system need not involve any language understanding. For instance, voice-controlled computers and VCRs are entering the market now. These do not involve natural language understanding in any general way. Rather, the words recognized are used as commands, much like the commands you send to a VCR using a remote control. Speech recognition is concerned only with identifying the words spoken from a

[Allen 1995 : Chapter 1 - Introduction / 6]

given speech signal, not with understanding how words are used to communicate. To be an understanding system, the speech recognizer would need to feed its input to a natural language understanding system, producing what is often called a spoken language understanding system.

With few exceptions, all the techniques discussed in this book are equally relevant for text-based and dialogue-based language understanding, and apply equally well whether the input is text, keyboard, or speech. The key characteristic of any understanding system is that it represents the meaning of sentences in some representation language that can be used later for further processing.

>> [back](#)

1.3 Evaluating Language Understanding Systems

As you can see, what counts as understanding might vary from application to application. If this is so, how can you tell if a system works? One obvious way to evaluate a system is to run the program and see how well it performs the task it was designed to do. If the program is meant to answer questions about a database of facts, you might ask it questions to see how good it is at producing the correct answers. If the system is designed to participate in simple conversations on a certain topic, you might try conversing with it. This is called black box evaluation because it evaluates system performance without looking inside to see how it works. While ultimately this method of evaluation may be the best test of a system's capabilities, it is problematic in the early stages of research because early evaluation results can be misleading. Sometimes the techniques that produce the best results in the short term will not lead to the best results in the long term. For instance, if the overall performance of all known systems in a given application is uniformly low, few conclusions can be drawn. The fact that one system was correct 50 percent of the time while another was correct only 40 percent of the time says nothing about the long-term viability of either approach. Only when the success rates become high, making a practical application feasible, can much significance be given to overall system performance measures.

An alternative method of evaluation is to identify various subcomponents of a system and then evaluate each one with appropriate tests. This is called glass box evaluation because you look inside at the structure of the system. The problem with glass box evaluation is that it requires some consensus on what the various components of a natural language system should be. Achieving such a consensus is an area of considerable activity at the present.

Despite the difficulties, evaluations are of crucial importance to the field, as very little can be concluded from examining a few examples that a particular system can handle. If you cannot interact freely with the program yourself but must depend on transcripts of others' interactions, you can easily be misled into overestimating a program's abilities. It is very hard for people not to attribute much more intelligence to the program than is actually present.

Consider a concrete example — the ELIZA program that was developed in the mid-1960s at MIT. This program must be one of the most popular AI

[Allen 1995 : Chapter 1 - Introduction / 7]

USER	ELIZA
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.

USER	ELIZA
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?

Figure 1.2 A dialogue with ELIZA

programs at all time, and versions of it exist for most machines, including most personal computers. ELIZA was never claimed to embody a theory of language comprehension and production, but it serves as an excellent example as its behavior initially seems impressive. The system plays the role of a therapist and, to obtain the best results, the user should correspondingly play the role of a patient. Figure 1.2 presents a transcript of this system in operation. Given this transcript, or even playing with the system yourself for a few minutes, ELIZA's performance certainly seems impressive.

Here is a simple description of how ELIZA works. There is a database of particular words that are called keywords. For each keyword, the system stores an integer, a pattern to match against the input, and a specification of the output. The algorithm is as follows: Given a sentence S, find a keyword in S whose pattern matches S. If there is more than one keyword, pick the one with the highest integer value. Use the output specification that is associated with this keyword to generate the next sentence. If there are no keywords, generate an innocuous continuation statement, such as "*Tell me more*" or "*Go on*".

Figure 1.3 shows a fragment of a database of keywords. In this database a pattern consists of words and variables. The prefix ? before a letter indicates a variable, which can match any sequence of words. For example, the pattern

?X are you ?Y

would match the sentence "*Why are you looking at me?*", where the variable ?X matches "*Why*" and "?Y" matches "*looking at me*". The output specification may also use the same variables. In this case, ELIZA inserts the words that match the variables in the input into the output after making some minor changes in the

[Allen 1995 : Chapter 1 - Introduction / 8]

Word	Rank	Pattern	Outputs
alike	10	?X	In what way? What resemblance do you see?
are	3	?X are you ?Y	Would you prefer it if I weren't ?Y?

Word	Rank	Pattern	Outputs
	3	?X are ?Y	What if they were not ?Y?
always	5	?X	Can you think of a specific example? When? Really, always?
what	2	?X	Why do you ask? Does that interest you?

Figure 1.3 Sample data from ELIZA

pronouns (for example, replacing "me" with "you"). Thus, for the pattern above, if the output specification is

Would you prefer it if I weren't ?Y?

the rule would generate a response "*Would you prefer it if I weren't looking at you?*" When the database lists multiple output specifications for a given pattern, ELIZA selects a different one each time a keyword rule is used, thereby preventing unnatural repetition in the conversation. Using these rules, you can see how ELIZA produced the first two exchanges in the conversation in Figure 1.2. ELIZA generated the first response from the first output of the keyword "alike" and the second response from the first output of the keyword "always".

This description covers all of the essential points of the program. You will probably agree that the program does not understand the conversation it is participating in. Rather, it is a collection of tricks. Why then does ELIZA appear to function so well? There are several reasons. Perhaps the most important reason is that, when people hear or read a sequence of words that they understand as a sentence, they attribute meaning to the sentence and assume that the person (or machine) that produced the sentence actually intended that meaning. People are extremely good at distinguishing word meanings and interpreting sentences to fit the context. Thus ELIZA appears to be intelligent because you use your own intelligence to make sense of what it says.

Other crucial characteristics of the conversational setting also aid in sustaining the illusion of intelligence. For instance, the system does not need any world knowledge because it never has to make a claim, support an argument, or answer a question. Rather, it simply asks a series of questions. Except in a patient-therapist situation, this would be unacceptable. ELIZA evades all direct questions by responding with another question, such as "*Why do you ask?*" There is no way to force the program to say something concrete about any topic.

[Allen 1995 : Chapter 1 - Introduction / 9]

Even in such a restricted situation, however, it is relatively easy to demonstrate that the program does not understand. It sometimes produces completely off-the-wall responses. For instance, if you say "*Necessity is the mother of invention*", it might respond with "*Tell me more about your family*", based on its pattern for the word "*mother*". In addition, since ELIZA has no knowledge about the structure of language, it accepts gibberish just as readily as valid sentences. If you enter "*Green the adzabak are the a ran four*", ELIZA will respond with something like "*What if they were not the a ran four?*" Also, as a conversation progresses, it becomes obvious that the program does not retain any of the content in the conversation. It begins to ask questions that are inappropriate in light of earlier exchanges, and its responses in general begin to show a lack of focus. Of course, if you are not able to play with the program and must depend only on transcripts of conversations by others, you would have no way of detecting these flaws, unless they are explicitly mentioned.

Suppose you need to build a natural language program for a certain application in only six months. If you start to construct a general model of language understanding, it will not be completed in that time frame and so will perform miserably on the tests. An ELIZA-like system, however, could easily produce behavior like that previously discussed with less than a few months of programming and will appear to far outperform the other system in testing. The differences will be especially marked if the test data only includes typical domain interactions that are not designed to test the limits of the system. Thus, if we take short-term performance as our only criteria of progress, everyone will build and fine-tune ELIZA-style systems, and the field will not progress past the limitations of the simple approach.

To avoid this problem, either we have to accept certain theoretical assumptions about the architecture of natural language systems and develop specific evaluation measures for different components, or we have to discount overall evaluation results until some reasonably high level of performance is obtained. Only then will cross-system comparisons begin to reflect the potential for long-term success in the field.

>> [back](#)

1.4 The Different Levels of Language Analysis

A natural language-system must use considerable knowledge about the structure of the language itself, including what the words are, how words combine to form sentences, what the words mean, how word meanings contribute to sentence meanings, and so on. However, we cannot completely account for linguistic behavior without also taking into account another aspect of what makes humans intelligent — their general world knowledge and their reasoning abilities. For example, to answer questions or to participate in a conversation, a person not only must know a lot about the structure of the language being used, but also must know about the world in general and the conversational setting in particular.

[Allen 1995 : Chapter 1 - Introduction / 10]

The following are some of the different forms of knowledge relevant for natural language understanding:

Phonetic and phonological knowledge - concerns how words are related to the sounds that realize them. Such knowledge is crucial for speech-based systems and is discussed in more detail in Appendix C.

Morphological knowledge - concerns how words are constructed from more basic meaning units called morphemes. A morpheme is the primitive unit of meaning in a language (for example, the meaning of the word "*friendly*" is derivable from the meaning of the noun "*friend*" and the suffix "*-ly*", which transforms a noun into an adjective).

Syntactic knowledge - concerns how words can be put together to form correct sentences and determines what structural role each word plays in the sentence and what phrases are subparts of what other phrases.

Semantic knowledge - concerns what words mean and how these meanings combine in sentences to form sentence meanings. This is the study of context-independent meaning - the meaning a sentence has regardless of the context in which it is used.

Pragmatic knowledge - concerns how sentences are used in different situations and how use affects the interpretation of the sentence.

Discourse knowledge-concerns how the immediately preceding sentences affect the interpretation of the next sentence. This information is especially important for interpreting pronouns and for interpreting the temporal aspects of the information conveyed.

World knowledge - includes the general knowledge about the structure of the world that language users must have in order to, for example, maintain a conversation. It includes what each language user must know about the other user's beliefs and goals.

These definitions are imprecise and are more characteristics of knowledge than actual distinct classes of knowledge. Any particular fact might include aspects from several different levels, and an algorithm might need to draw from several different levels simultaneously. For teaching purposes, however, this book is organized into three parts, each describing a set of techniques that naturally cluster together. Part I focuses on syntactic and morphological processing, Part II focuses on semantic processing, and Part III focuses on contextual effects in general, including pragmatics, discourse, and world knowledge.

[Allen 1995 : Chapter 1 - Introduction / 11]

BOX 1.2 Syntax, Semantics, and Pragmatics

The following examples may help you understand the distinction between syntax, semantics, and pragmatics. Consider each example as a candidate for the initial sentence of this book, which you know discusses natural language processing:

1. Language is one of the fundamental aspects of human behavior and is a crucial component of our lives.
2. Green frogs have large noses.

3. Green ideas have large noses.
4. Large have green ideas nose.

Sentence 1 appears to be a reasonable start (I hope!). It agrees with all that is known about syntax, semantics, and pragmatics. Each of the other sentences violates one or more of these levels. Sentence 2 is well-formed syntactically and semantically, but not pragmatically. It fares poorly as the first sentence of the book because the reader would find no reason for using it. But however bad sentence 2 would be as a start, sentence 3 is much worse. Not only is it obviously pragmatically ill-formed, it is also semantically ill-formed. To see this, consider that you and I could argue about whether sentence 2 is true or not, but we cannot do so with sentence 3. I cannot affirm or deny sentence 3 in coherent conversation. However, the sentence does have some structure, for we can discuss what is wrong with it: Ideas cannot be green and, even if they could, they certainly cannot have large noses. Sentence 4 is even worse. In fact, it is unintelligible, even though it contains the same words as sentence 3. It does not even have enough structure to allow you to say what is wrong with it. Thus it is syntactically ill-formed. Incidentally, there are cases in which a sentence may be pragmatically well-formed but not syntactically well-formed. For example, if I ask you where you are going and you reply "I go store", the response would be understandable even though it is syntactically ill-formed. Thus it is at least pragmatically well-formed and may even be semantically well-formed.

>> [back](#)

1.5 Representations and Understanding

As previously stated, a crucial component of understanding involves computing a representation of the meaning of sentences and texts. Without defining the notion of representation, however, this assertion has little content. For instance, why not simply use the sentence itself as a representation of its meaning? One reason is that most words have multiple meanings, which we will call senses. The word "*cook*", for example, has a sense as a verb and a sense as a noun; "*dish*" has multiple senses as a noun as well as a sense as a verb; and "*still*" has senses as a noun, verb, adjective, and adverb. This ambiguity would inhibit the system from making the appropriate inferences needed to model understanding. The disambiguation problem appears much easier than it actually is because people do not generally notice ambiguity. While a person does not seem to consider each of the possible

[Allen 1995 : Chapter 1 - Introduction / 12]

senses of a word when understanding a sentence, a program must explicitly consider them one by one.

To represent meaning, we must have a more precise language. The tools to do this come from mathematics and logic and involve the use of formally specified representation languages. Formal languages are specified from very simple building blocks. The most fundamental is the notion of an atomic symbol which is distinguishable from any other atomic symbol simply based on how it is written. Useful representation languages have the following two properties:

- The representation must be precise and unambiguous. You should be able to express every distinct reading of a sentence as a distinct formula in the representation.
- The representation should capture the intuitive structure of the natural language sentences that it represents. For example, sentences that appear to be structurally similar should have similar structural representations, and the meanings of two sentences that are paraphrases of each other should be closely related to each other.

Several different representations will be used that correspond to some of the levels of analysis discussed in the last section. In particular, we will develop formal languages for expressing syntactic structure, for context-independent word and sentence meanings, and for expressing general world knowledge.

Syntax: Representing Sentence Structure

The syntactic structure of a sentence indicates the way that words in the sentence are related to each other. This structure indicates how the words are grouped together into phrases, what words modify what other words, and what words are of central importance in the sentence. In addition, this structure may identify the types of relationships that exist between phrases and can store other information about the particular sentence structure that may be needed for later processing. For example, consider the following sentences:

1. *John sold the book to Mary.*
2. *The book was sold to Mary by John.*

These sentences share certain structural properties. In each, the noun phrases are "*John*", "*Mary*", and "*the book*", and the act described is some selling action. In other respects, these sentences are significantly different. For instance, even though both sentences are always either true or false in the exact same situations, you could only give sentence 1 as an answer to the question "*What did John do for Mary?*" Sentence 2 is a much better continuation of a sentence beginning with the phrase "*After it fell in the river*", as sentences 3 and 4 show. Following the standard convention in linguistics, this book will use an asterisk (*) before any example of an ill-formed or questionable sentence.

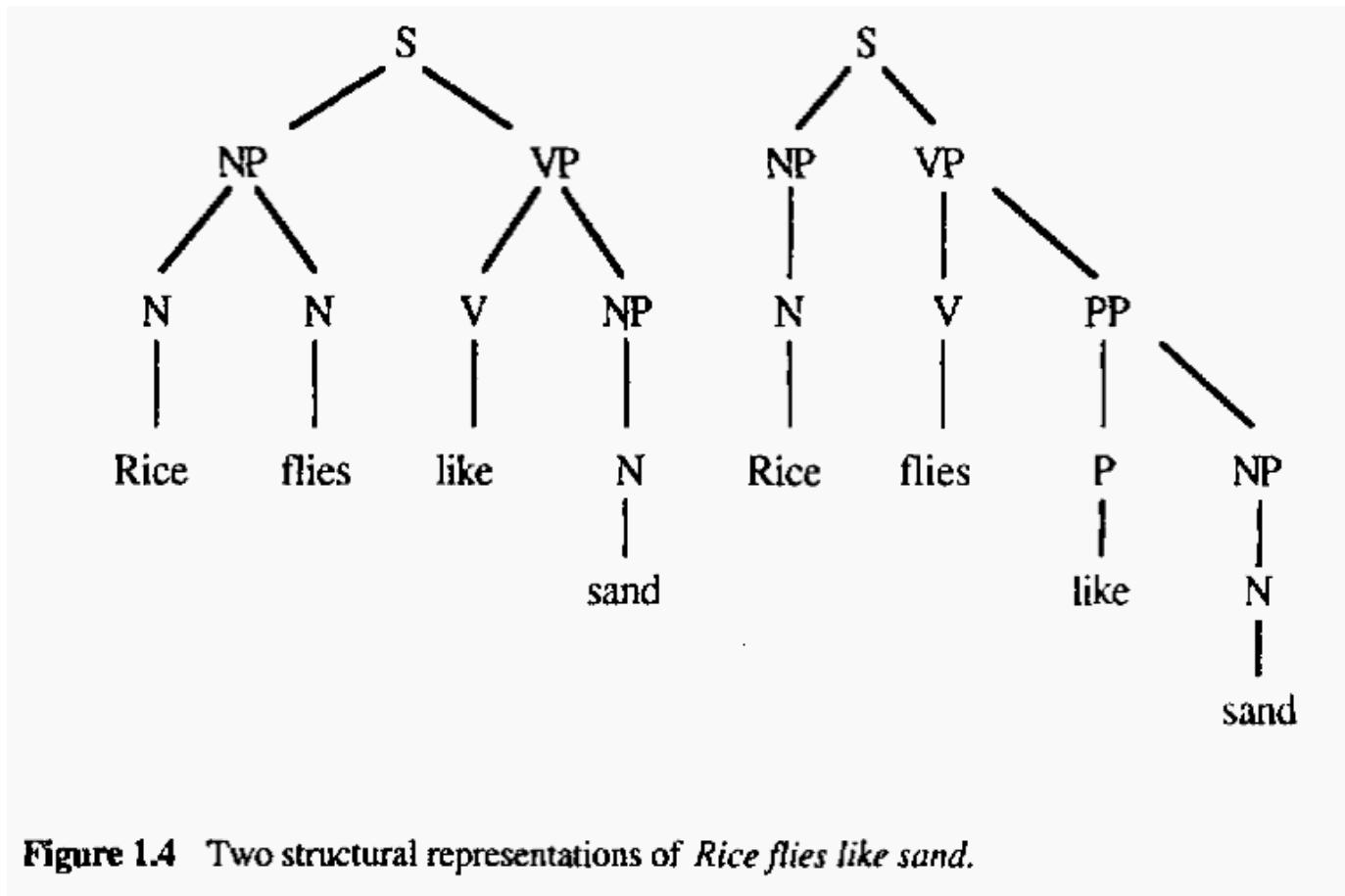


Figure 1.4 Two structural representations of *Rice flies like sand*.

Figure 1.4 Two structural representations of "*Rice flies like sand*".

3. *After it fell in the river, John sold Mary the book.
4. After it fell in the river, the book was sold to Mary by John.

Many other structural properties can be revealed by considering sentences that are not well-formed. Sentence 5 is ill-formed because the subject and the verb do not agree in number (the subject is singular and the verb is plural), while 6 is ill-formed because the verb *put* requires some modifier that describes where John put the object.

5. *John are in the corner.
6. *John put the book.

Making judgments on grammaticality is not a goal in natural language understanding. In fact, a robust system should be able to understand ill-formed sentences whenever possible. This might suggest that agreement checks

can be ignored, but this is not so. Agreement checks are essential for eliminating potential ambiguities. Consider sentences 7 and 8, which are identical except for the number feature of the main verb, yet represent two quite distinct interpretations.

7. *flying planes are dangerous.*

8. *flying planes is dangerous.*

If you did not check subject-verb agreement, these two sentences would be indistinguishable and ambiguous. You could find similar examples for every syntactic feature that this book introduces and uses.

Most syntactic representations of language are based on the notion of context-free grammars, which represent sentence structure in terms of what phrases are subparts of other phrases. This information is often presented in a tree form, such as the one shown in Figure 1.4, which shows two different structures

[Allen 1995 : Chapter 1 - Introduction / 14]

for the sentence "*Rice flies like sand*". In the first reading, the sentence is formed from a noun phrase (NP) describing a type of fly' rice flies, and a verb phrase (VP) that asserts that these flies like sand. In the second structure, the sentence is formed from a noun phrase describing a type of substance, rice, and a verb phrase stating that this substance flies like sand (say, if you throw it). The two structures also give further details on the structure of the noun phrase and verb phrase and identify the part of speech for each word. In particular, the word "*like*" is a verb (V) in the first reading and a preposition (P) in the second.

The Logical Form

The structure of a sentence doesn't reflect its meaning, however. For example, the NP "*the catch*" can have different meanings depending on whether the speaker is talking about a baseball game or a fishing expedition. Both these interpretations have the same syntactic structure, and the different meanings arise from an ambiguity concerning the sense of the word "*catch*". Once the correct sense is identified, say the fishing sense, there still is a problem in determining what fish are being referred to. The intended meaning of a sentence depends on the situation in which the sentence is produced. Rather than combining all these problems, this book will consider each one separately. The division is between context-independent meaning and context-dependent meaning. The fact that "*catch*" may refer to a baseball move or the results of a fishing expedition is knowledge about English and is independent of the situation in which the word is used. On the other hand, the fact that a particular noun

phrase "*the catch*" refers to what Jack caught when fishing yesterday is contextually dependent. The representation of the context-independent meaning of a sentence is called its logical form.

The logical form encodes possible word senses and identifies the semantic relationships between the words and phrases. Many of these relationships are often captured using an abstract set of semantic relationships between the verb and its NPs. In particular, in both sentences 1 and 2 previously given, the action described is a selling event, where "*John*" is the seller, "*the book*" is the object being sold, and "*Mary*" is the buyer. These roles are instances of the abstract semantic roles AGENT, THEME, and TO-POSS (for final possessor), respectively.

Once the semantic relationships are determined, some word senses may be impossible and thus eliminated from consideration. Consider the sentence

9. *Jack invited Mary to the Halloween ball.*

The word "*ball*", which by itself is ambiguous between the plaything that bounces and the formal dance event, can only take the latter sense in sentence 9, because the verb "*invite*" only makes sense with this interpretation. One of the key tasks in semantic interpretation is to consider what combinations of the individual word meanings can combine to create coherent sentence meanings. Exploiting such

[Allen 1995 : Chapter 1 - Introduction / 15]

interconnections between word meanings can greatly reduce the number of possible word senses for each word in a given sentence.

The Final Meaning Representation

The final representation needed is a general knowledge representation (KR), which the system uses to represent and reason about its application domain. This is the language in which all the specific knowledge based on the application is represented. The goal of contextual interpretation is to take a representation of the structure of a sentence and its logical form, and to map this into some expression in the KR that allows the system to perform the appropriate task in the domain. In a question-answering application, a question might map to a database query, in a story-understanding application, a sentence might map into a set of expressions that represent the situation that the sentence describes.

For the most part, we will assume that the first-order predicate calculus (FOPC) is the final representation language because it is relatively well known, well studied, and is precisely defined. While some inadequacies of FOPC will be examined later, these inadequacies are not relevant for most of the issues to be discussed.

>> [back](#)

1.6 The Organization of Natural Language Understanding Systems

This book is organized around the three levels of representation just discussed: syntactic structure, logical form, and the final meaning representation. Separating the problems in this way will allow you to study each problem in depth without worrying about other complications. Actual systems are usually organized slightly differently, however. In particular, Figure 1.5 shows the organization that this book assumes.

As you can see, there are interpretation processes that map from one representation to the other. For instance, the process that maps a sentence to its syntactic structure and logical form is called the parser. It uses knowledge about word and word meanings (the lexicon) and a set of rules defining the legal structures (the grammar) in order to assign a syntactic structure and a logical form to an input sentence. An alternative organization could perform syntactic processing first and then perform semantic interpretation on the resulting structures. Combining the two, however, has considerable advantages because it leads to a reduction in the number of possible interpretations, since every proposed interpretation must simultaneously be syntactically and semantically well formed. For example, consider the following two sentences:

10. Visiting relatives can be trying.

11. Visiting museums can be trying.

These two sentences have identical syntactic structure, so both are syntactically ambiguous. In sentence 10, the subject might be relatives who are visiting you or

[Allen 1995 : Chapter 1 - Introduction / 16]

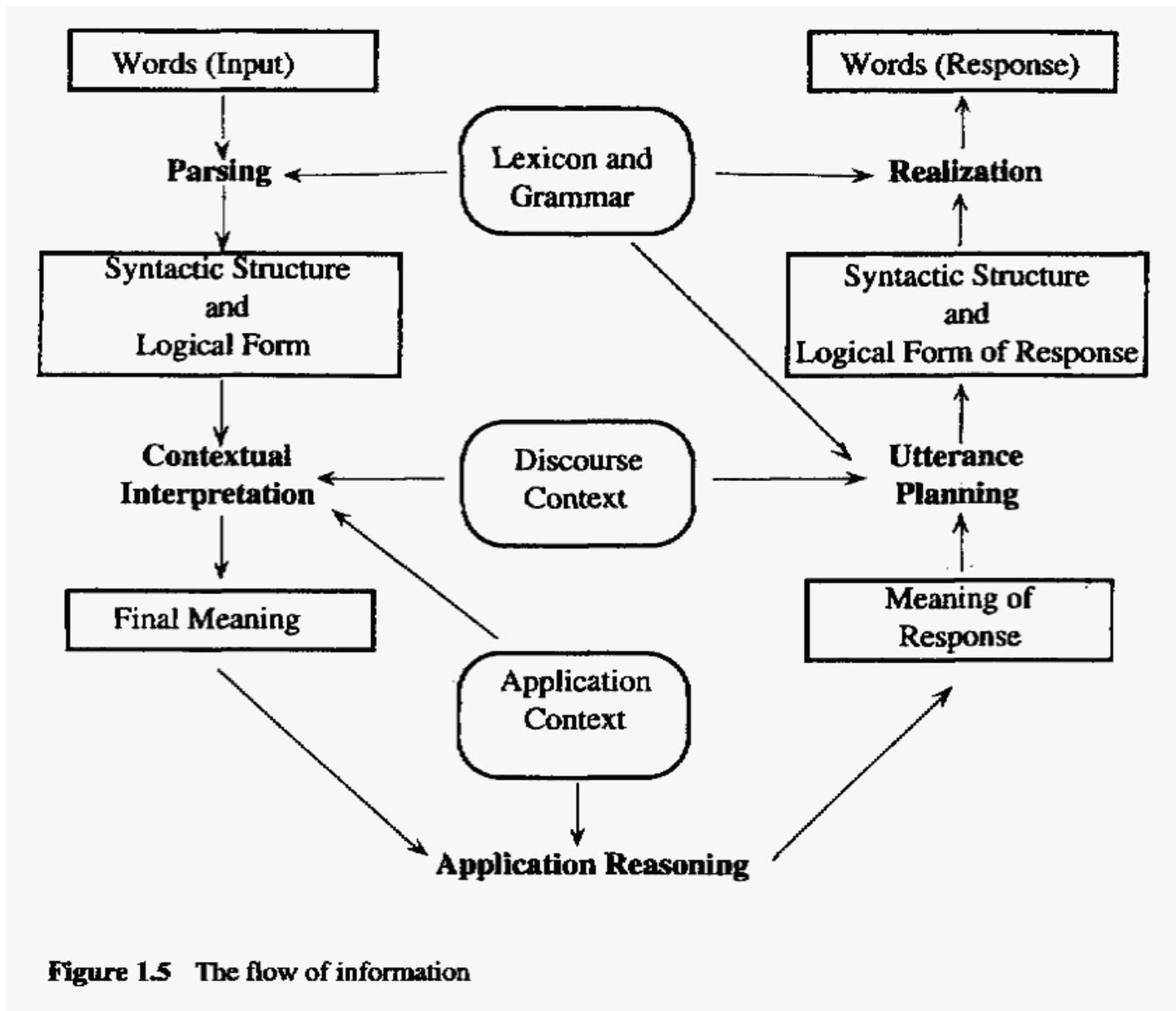


Figure 1.5 The flow of information

Figure 1.5 The flow of information

the event of you visiting relatives. Both of these alternatives are semantically valid, and you would need to determine the appropriate sense by using the contextual mechanism. However, sentence 11 has only one possible semantic interpretation, since museums are not objects that can visit other people; rather they must be visited. In a system with separate syntactic and semantic processing, there would be two syntactic interpretations of sentence 11, one of which the semantic interpreter would eliminate later. If syntactic and semantic processing are combined, however, the system will be able to detect the semantic anomaly as soon as it interprets the phrase "*visiting museums*", and thus will never build the incorrect syntactic structure in the first place. While the savings here seem small, in a realistic application a reasonable sentence may have hundreds of possible syntactic structures, many of which are semantically anomalous.

Continuing through Figure 1.5, the process that transforms the syntactic structure and logical form into a final meaning representation is called contextual processing. This process includes issues such as identifying the objects referred to by noun phrases such as definite descriptions (for example, "*the man*") and pronouns, the analysis of the temporal aspects of the new information conveyed by the sentence, the identification of the speaker's intention (for example, whether "*Can you lift that rock*" is a yes/no question or a request), as well as all the

[Allen 1995 : Chapter 1 - Introduction / 17]

inferential processing required to interpret the sentence appropriately within the application domain. It uses knowledge of the discourse context (determined by the sentences that preceded the current one) and knowledge of the application to produce a final representation.

The system would then perform whatever reasoning tasks are appropriate for the application. When this requires a response to the user, the meaning that must be expressed is passed to the generation component of the system. It uses knowledge of the discourse context, plus information on the grammar and lexicon, to plan the form of an utterance, which then is mapped into words by a realization process. Of course, if this were a spoken language application, the words would not be the final input and output, but rather would be the output of a speech recognizer and the input to a speech synthesizer, as appropriate.

While this text focuses primarily on language understanding, notice that the same levels of knowledge are also used for the generation task as well. For instance, knowledge of syntactic structure is encoded in the grammar. This grammar can be used either to identify the structure of a given sentence or to realize a structure as a sequence of words. A grammar that supports both processes is called a bidirectional grammar. While most researchers agree that bidirectional grammars are the preferred model, in actual practice grammars are often tailored for the understanding task or the generation task. This occurs because different issues are important for each task, and

generally any given researcher focuses just on the problems related to their specific task. But even when the actual grammars differ between understanding and generation, the grammatical formalisms used remain the same.

>> [back](#)

Summary

This book describes computational theories of natural language understanding. The principal characteristic of understanding systems is that they compute representations of the meanings of sentences and use these representations in reasoning tasks. Three principal levels of representation were introduced that correspond to the three main subparts of this book. Syntactic processing is concerned with the structural properties of sentences; semantic processing computes a logical form that represents the context-independent meaning of the sentence; and contextual processing connects language to the application domain.

>> [back](#)

Related Work and Further Readings

A good idea of work in the field can be obtained by reading two articles in Shapiro (1992), under the headings "Computational Linguistics" and "Natural Language Understanding". There are also articles on specialized subareas such as machine translation, natural language interfaces, natural language generation, and so on. Longer surveys on certain areas are also available. Slocum (1985) gives a

[Allen 1995 : Chapter 1 - Introduction / 18]

survey of machine translation, and Perrault and Grosz (1986) give a survey of natural language interfaces.

You can find a description of the ELIZA program that includes the transcript of the dialogue in Figure 1.2 in Weizenbaum (1966). The basic technique of using template matching was developed further in the PARRY system, as described in the paper by Colby in Schank and Colby (1973). That same book also contains descriptions of early natural language systems, including those by Winograd and by Schank. Another important

early system is the LUNAR system, an overview of which can be found in Woods (1977). For another perspective on the AI approach to natural language, refer to the introduction in Winograd (1983).

>> [back](#)

Exercises for Chapter 1

1. (easy) Define a set of data rules for ELIZA that would generate the first seven exchanges in the conversation in Figure 1.2.
2. (easy) Discover all of the possible meanings of the following sentences by giving a paraphrase of each interpretation. For each sentence, identify whether the different meanings arise from structural ambiguity, semantic ambiguity, or pragmatic ambiguity.
 - a. Time flies like an arrow.
 - b. He drew one card.
 - c. Mr. Spook was charged with illegal alien recruitment.
 - d. He crushed the key to my heart.
3. (*easy*) Classify these sentences along each of the following dimensions, given that the person uttering the sentence is responding to a complaint that the car is too cold: (i) syntactically correct or not; (ii) semantically correct or not; (iii) pragmatically correct or not.

- a. The heater are on.
 - b. The tires are brand new.
 - c. Too many windows eat the stew.
4. (*medium*) Implement an ELIZA program that can use the rules that you developed in Exercise 1 and run it for that dialogue. Without adding any more rules, what does your program do on the next few utterances in the conversation in Figure 1.2? How does the program do if you run it in a different context - say, a casual conversation at a bar?

[Allen 1995 : Chapter 1 - Introduction / 19]

Part I: Syntactic Processing

[Allen 1995 : Chapter 2 - An Outline of English Syntax / 20]

As discussed in the introduction, this book divides the task of understanding sentences into three stages. Part I of the book discusses the first stage, syntactic processing. The goal of syntactic processing is to determine the structural components of sentences. It determines, for instance, how a sentence is broken down into phrases, how those phrases are broken down into sub-phrases, and so on, all the way down to the actual structure of the words used. These structural relationships are crucial for determining the meaning of sentences using the techniques described in Parts II and III.

There are two major issues discussed in Part I. The first issue concerns the formalism that is used to specify what sentences are possible in a language. This information is specified by a set of rules called a grammar. We will be concerned both with the general issue of what constitutes good formalisms for writing grammars for natural languages, and with the specific issue of what grammatical rules provide a good account of English syntax. The second issue concerns how to determine the structure of a given sentence once you know the grammar for the language. This process is called parsing. There are many different algorithms for parsing, and this book will consider a sampling of the techniques that are most influential in the field.

Chapter 2 provides a basic background to English syntax for the reader who has not studied linguistics. It introduces the key concepts and distinctions that are common to virtually all syntactic theories. Chapter 3 introduces several formalisms that are in common use for specifying grammars and describes the basic parsing algorithms in detail. Chapter 4 introduces the idea of features, which extend the basic grammatical formalisms and allow many aspects of natural languages to be captured concisely. Chapter 5 then describes some of the more difficult aspects of natural languages, especially the treatment of questions, relative clauses, and other forms of movement phenomena. It shows how the feature systems can be extended so that they can handle these complex sentences. Chapter 6 discusses issues relating to ambiguity resolution. Some techniques are aimed at developing more efficient representations for storing multiple interpretations, while others are aimed at using local information to choose between alternative interpretations while the parsing is in progress. Finally, Chapter 7 discusses a relatively new area of research that uses statistical information derived from analyzing large databases of sentences. This information can be used to identify the most likely classes for ambiguous words and the most likely structural analyses for structurally ambiguous sentences.

>> [back](#)

Allen 1995: Natural Language Understanding

	<u>Contents</u>	<u>Preface</u>	<u>Introduction</u>	
<u>previous chapter</u>	<u>Part I Syntactic Processing</u>	<u>Part II Semantic Interpretation</u>	<u>Part III - Context / World Knowledge</u>	<u>next chapter</u>
	<u>Appendices</u>	<u>Bibliography</u>		
	<u>Summaries</u>	<u>Further Readings</u>	<u>Exercises</u>	

Allen, James:

Natural Language Understanding.

University of Rochester

New York / Ontario / Wokingham, U.K. / Amsterdam / Bonn / Sydney / Singapore / Tokyo / Madrid / San Juan 1995

The Benjamin/Cummings Publishing Company, Inc.

Redwood City, California • Menlo Park, California Reading, Massachusetts

New York • Don Mills, Ontario • Wokingham, U.K. • Amsterdam • Bonn • Sydney • Singapore • Tokyo • Madrid •
San Juan

ISBN: 0-8053-0334-0

Contents

Preface

Introduction

[Chapter 1 - Introduction to Natural Language Understanding](#)

Part I - Syntactic Processing

[Chapter 2 - Linguistic Background: An Outline of English Syntax](#)

[Chapter 3 - Grammars and Parsing](#)

[Chapter 4 - Features and Augmented Grammars](#)

[Chapter 5 - Grammars for Natural Language](#)

[Chapter 6 - Toward Efficient Parsing](#)

[Chapter 7 - Ambiguity Resolution: Statistical Methods](#)

Part II - Semantic Interpretation

[Chapter 8 - Semantics and Logical Form](#)

Preface

[Chapter 9 - Linking Syntax and Semantics](#)

[Chapter 10 - Ambiguity Resolution](#)

[Chapter 11 - Other Strategies for Semantic Interpretation](#)

[Chapter 12 - Scoping and the Interpretation of Noun Phrases](#)

Part III - Context and World Knowledge

Chapter 13 - Knowledge Representation and Reasoning

Chapter 14 -Local Discourse Context and Reference

Chapter 15 - Using World Knowledge

Chapter 16 - Discourse Structure

Chapter 17 - Defining a Conversational Agent

Appendix A - An Introduction to Logic and Model-Theoretic Semantics

Appendix B - Symbolic Computation

Appendix C - Speech Recognition and Spoken Language

[Preface](#)

[Bibliography](#)

[Index](#)

[Chapter 1 - Introduction to Natural Language Understanding](#)

^

[1.1 The Study of Language 1](#)

[1.2 Applications of Natural Language Understanding 3](#)

[1.3 Evaluating Language Understanding Systems 6](#)

[1.4 The Different Levels of Language Analysis 9](#)

[1.5 Representations and Understanding 11](#)

[1.6 The Organization of Natural Language Understanding Systems 15](#)

PART I - SYNTACTIC PROCESSING

Chapter 2 - Linguistic Background: An Outline of English Syntax

^

2.1 Words 23

2.2 The Elements of Simple Noun Phrases 25

2.3 Verb Phrases and Simple Sentences 28

2.4 Noun Phrases Revisited 33

2.5 Adjective Phrases 35

2.6 Adverbial Phrases 35

Chapter 3 - Grammars and Parsing

^

3.1 Grammars and Sentence Structure 41

3.2 What Makes a Good Grammar 44

3.3 A Top-Down Parser 47

3.4 A Bottom-Up Chart Parser 53

3.5 Transition Network Grammars 61

o 3.6 Top-Down Chart Parsing 65

o 3.7 Finite State Models and Morphological Processing 70

o 3.8 Grammars and Logic Programming 72

(a optional topic)

Chapter 4 - Features and Augmented Grammars

^

[4.1 Feature Systems and Augmented Grammars 83](#)

[4.2 Some Basic Feature Systems for English 86](#)

[4.3 Morphological Analysis and the Lexicon 90](#)

[4.4 A Simple Grammar Using Features 94](#)

[4.5 Parsing with Features 98](#)

[o 4.6 Augmented Transition Networks 101](#)

[o 4.7 Definite Clause Grammars 106](#)

[o 4.8 Generalized Feature Systems and Unification Grammars 109](#)

Chapter 5 - Grammars for Natural Language

^
—

5.1 Auxiliary Verbs and Verb Phrases 123

5.2 Movement Phenomena in Language 127

5.3 Handling Questions in Context-Free Grammars 132

5.4 Relative Clauses 141

5.5 The Hold Mechanism in ATNs 144

5.6 Gap Threading 148

Chapter 6 - Toward Efficient Parsing

^
—

6.1 Human Preferences in Parsing 159

Chapter 6 - Toward Efficient Parsing

^

6.2 Encoding Uncertainty: Shift-Reduce Parsers 163

6.3 A Deterministic Parser 170

6.4 Techniques for Efficient Encoding of Ambiguity 176

6.5 Partial Parsing 180

Chapter 7 - Ambiguity Resolution: Statistical Methods

^

7.1 Basic Probability Theory 189

7.2 Estimating Probabilities 192

7.3 Part-of-Speech Tagging 195

7.4 Obtaining Lexical Probabilities 204

Chapter 7 - Ambiguity Resolution: Statistical Methods

^

7.5 Probabilistic Context-Free Grammars 209

7.6 Best-First Parsing 213

7.7 A Simple Context-Dependent Best-First Parser 216

PART II - SEMANTIC INTERPRETATION

Chapter 8 - Semantics and Logical Form

^

8.1 Semantics and Logical Form 227

Chapter 8 - Semantics and Logical Form

^
—

8.2 Word Senses and Ambiguity 231

8.3 The Basic Logical Form Language 233

8.4 Encoding Ambiguity in the Logical Form 238

8.5 Verbs and States in Logical Form 241

8.6 Thematic Roles 244

8.7 Speech Acts and Embedded Sentences 250

8.8 Defining Semantic Structure: Model Theory 251

Chapter 9 - Linking Syntax and Semantics

^
—

9.1 Semantic Interpretation and Compositionality 263

Chapter 9 - Linking Syntax and Semantics

^
—

[9.2 A Simple Grammar and Lexicon with Semantic Interpretation 267](#)

[9.3 Prepositional Phrases and Verb Phrases 271](#)

[9.4 Lexicalized Semantic Interpretation and Semantic Roles 275](#)

[9.5 Handling Simple Questions 280](#)

[9.6 Semantic Interpretation Using Feature Unification 283](#)

[o 9.7 Generating Sentences from Logical Form 286](#)

Chapter 10 - Ambiguity Resolution

^
—

10.1 Selectional Restrictions 295

10.2 Semantic Filtering Using Selectional Restrictions 302

Chapter 10 - Ambiguity Resolution

^

10.3 Semantic Networks 305

10.4 Statistical Word Sense Disambiguation 310

10.5 Statistical Semantic Preferences 314

10.6 Combining Approaches to Disambiguation 318

Chapter 11 - Other Strategies for Semantic Interpretation

^

11.1 Grammatical Relations 328

11.2 Semantic Grammars 332

11.3 Template Matching 334

11.4 Semantically Driven Parsing Techniques 341

Chapter 12 - Scoping and the Interpretation of Noun Phrases

^

12.1 Scoping Phenomena 351

12.2 Definite Descriptions and Scoping 359

12.3 A Method for Scoping While Parsing 360

12.4 Co-Reference and Binding Constraints 366

12.5 Adjective Phrases 372

12.6 Relational Nouns and Nominalizations 375

12.7 Other Problems in Semantics 378

PART III - CONTEXT AND WORLD KNOWLEDGE

Chapter 13 - Knowledge Representation and Reasoning

^

13.1 Knowledge Representation 392

13.2 A Representation Based on FOPC 397

13.3 Frames: Representing Stereotypical Information 400

13.4 Handling Natural Language Quantification 404

◦ 13.5 Time and Aspectual Classes of Verbs 406

◦ 13.6 Automating Deduction in Logic-Based Representations 410

◦ 13.7 Procedural Semantics and Question Answering 414

◦ 13.8 Hybrid Knowledge Representations 419

Chapter 14 -Local Discourse Context and Reference



14.1 Defining Local Discourse Context and Discourse Entities 429

14.2 A Simple Model of Anaphora Based on History Lists 433

14.3 Pronouns and Centering 435

14.4 Definite Descriptions 440

◦ 14.5 Definite Reference and Sets 445

14.6 Ellipsis 449

14.7 Surface Anaphora 455

Chapter 15 - Using World Knowledge



Chapter 15 - Using World Knowledge



15.1 Using World Knowledge: Establishing Coherence 465

15.2 Matching Against Expectations 466

15.3 Reference and Matching Expectations 471

15.4 Using Knowledge About Action and Causality 473

15.5 Scripts: Understanding Stereotypical Situations 477

15.6 Using Hierarchical Plans 480

o 15.7 Action-Effect-Based Reasoning 483

o 15.8 Using Knowledge About Rational Behavior 490

Chapter 16 - Discourse Structure



16.1 The Need for Discourse Structure 503

Chapter 16 - Discourse Structure



16.2 Segmentation and Cue Phrases 504

16.3 Discourse Structure and Reference 510

16.4 Relating Discourse Structure and Inference 512

16.5 Discourse Structure, Tense, and Aspect 517

16.6 Managing the Attentional Stack 524

16.7 An Example 530

Chapter 17 - Defining a Conversational Agent



17.1 What's Necessary to Build a Conversational Agent? 541

17.2 Language as a Multi-Agent Activity 543

Chapter 17 - Defining a Conversational Agent

^

17.3 Representing Cognitive States: Beliefs 545

17.4 Representing Cognitive States: Desires, Intentions, and Plans 551

17.5 Speech Acts and Communicative Acts 554

17.6 Planning Communicative Acts 557

17.7 Communicative Acts and the Recognition of Intention 561

17.8 The Source of Intentions in Dialogue 564

17.9 Recognizing Illocutionary Acts 567

17.10 Discourse-Level Planning 570

Appendices

APPENDIX A - An Introduction to Logic and Model-Theoretic Semantics



A.1 Logic and Natural Language 579

A.2 Model-Theoretic Semantics 584

A.3 A Semantics for FOPC: Set-Theoretic Models 588

APPENDIX B - Symbolic Computation



B.1 Symbolic Data Structures 595

B.2 Matching 598

APPENDIX B - Symbolic Computation

[^](#)

B.3 Search Algorithms 600

B.4 Logic Programming 603

B.5 The Unification Algorithm 604

APPENDIX C - Speech Recognition and Spoken Language

[^](#)

C.1 Issues in Speech Recognition 611

C.2 The Sound Structure of Language 613

C.3 Signal Processing 616

C.4 Speech Recognition 619

C.5 Speech Recognition and Natural Language Understanding 623

APPENDIX C - Speech Recognition and Spoken Language

 ^

C.6 Prosody and Intonation 625

>> [back](#)

Summaries

[Chapter 1 - Introduction to Natural Language Understanding](#)

[Chapter 2 - Linguistic Background: An Outline of English Syntax](#)

[Chapter 3 - Grammars and Parsing](#)

[Chapter 4 - Features and Augmented Grammars](#)

[Chapter 5 - Grammars for Natural Language](#)

[Chapter 6 - Toward Efficient Parsing](#)

[Chapter 7 - Ambiguity Resolution: Statistical Methods](#)

[Chapter 8 - Semantics and Logical Form](#)

[Chapter 9 - Linking Syntax and Semantics](#)

>> [back](#)

Related Work and Further Readings

[Chapter 1 - Introduction to Natural Language Understanding](#)

[Chapter 2 - Linguistic Background: An Outline of English Syntax](#)

[Chapter 3 - Grammars and Parsing](#)

[Chapter 4 - Features and Augmented Grammars](#)

[Chapter 5 - Grammars for Natural Language](#)

[Chapter 6 - Toward Efficient Parsing](#)

[Chapter 7 - Ambiguity Resolution: Statistical Methods](#)

[Chapter 8 - Semantics and Logical Form](#)

[Chapter 9 - Linking Syntax and Semantics](#)

>> [back](#)

Exercises

[Chapter 1 - Introduction to Natural Language Understanding](#)

[Chapter 2 - Linguistic Background: An Outline of English Syntax](#)

[Chapter 3 - Grammars and Parsing](#)

[Chapter 4 - Features and Augmented Grammars](#)

[Chapter 5 - Grammars for Natural Language](#)

[Chapter 6 - Toward Efficient Parsing](#)

[Chapter 7 - Ambiguity Resolution: Statistical Methods](#)

[Chapter 8 - Semantics and Logical Form](#)

[Chapter 9 - Linking Syntax and Semantics](#)

>> [back](#)

Allen 1995: Natural Language Understanding

	<u>Contents</u>	<u>Preface</u>	<u>Introduction</u>	
<u>previous chapter</u>	<u>Part I Syntactic Processing</u>	<u>Part II Semantic Interpretation</u>	<u>Part III - Context / World Knowledge</u>	<u>next chapter</u>
	<u>Appendices</u>	<u>Bibliography</u>		
	<u>Summaries</u>	<u>Further Readings</u>	<u>Exercises</u>	

Preface

The primary goal of this book is to provide a comprehensive, in-depth description of the theories and techniques used in the field of natural language understanding. To do this in a single book requires eliminating the idiosyncratic complexities of particular approaches and identifying the underlying concepts of the field as a whole. It also requires developing a small number of notations that can be used to describe a wide range of techniques. It is not possible to capture the range of specific notations that are used in the literature, and attempting to do so would mask the important ideas rather than illuminate them. This book also attempts to make as few assumptions about the background of the reader as possible. All issues and techniques are described in plain English whenever possible. As a result, any reader having some familiarity with the basic notions of programming will be able to understand the principal ideas and techniques used. There is enough detail in this book, however, to allow a sophisticated programmer to produce working systems for natural language understanding.

The book is intended both as textbook and as a general reference source for those interested in natural language processing. As a text, it can be used both in undergraduate and graduate courses in computer science or computational linguistics. As a reference source, it can be used by researchers in areas related to natural language, by developers building natural language systems, and by individuals who are simply interested in how computers can process language.

Work in natural language understanding requires background in a wide range of areas, most importantly computer science, linguistics, logic, psycholinguistics, and the philosophy of language. As very few people have all this background, this book introduces whatever background material is required, as needed. For those who have no familiarity with symbolic programming or logical formalisms, two appendices are provided that describe enough of the basic ideas to enable nonprogrammers to read the book. Background in linguistics and other areas is introduced as needed throughout the book.

The book is organized by problem area rather than by technique. This allows the overlap in different approaches to be presented once, and facilitates the comparison of the different techniques. For example, rather than having a chapter on context-free grammars, one on transition network grammars, and another on logic programming techniques, these three techniques are discussed in many chapters, each focusing on a specific set of problems. There is a chapter on basic parsing techniques, then another on using features, and another on handling movement phenomena in language. Each chapter lays out its problem, and then discusses how the problem is addressed in each approach. This way, the similarities and differences between the approaches can be clearly identified.

>> [back](#)

Allen 1995 : Natural Language Understanding

	<u>Contents</u>	<u>Preface</u>	<u>Introduction</u>	
<u>previous chapter</u>	<u>Part I Syntactic Processing</u>	<u>Part II Semantic Interpretation</u>	<u>Part III - Context / World Knowledge</u>	<u>next chapter</u>
	<u>Appendices</u>	<u>Bibliography</u>		
	<u>Summaries</u>	<u>Further Readings</u>	<u>Exercises</u>	

Part II: Semantic Interpretation

As discussed in the introduction, the task of determining the meaning of a sentence will be divided into two steps: first computing a context-independent notion of meaning, called the logical form, and then interpreting the logical form in context to produce the final meaning representation. Part II of the book concerns the first of these steps, which will be called semantic interpretation. Many actual systems do not make this division and use contextual information early in the processing. Nonetheless, the organization is very helpful for classifying the issues and for illustrating the different techniques.

A similar division is often made in linguistics, where the study of context-independent meaning is often called semantics, and the study of language in context is called pragmatics. Using the need for context as the test for dividing up problems, the following are some examples of issues that can be addressed in a context-independent way and hence will be addressed in Part II:

- eliminating possible word senses by exploiting context-independent structural constraints
- identifying the semantic roles that each word and phrase could play in the logical form, specifically identifying the predicate/argument and modification relations
- identifying co-reference restrictions derived from the structure of the sentence

Just as important, the following will *not* be studied in Part II but will be left for the discussion of contextual interpretation in Part III:

- determining the referents of noun phrases and other phrases
- selecting a single interpretation (and set of word senses) from those that are possible
- determining the intended use of each utterance

Chapter 8 discusses the distinction between logical form and the final meaning representation in more detail, and develops a logical form language that is used throughout the rest of the book. Chapter 9 then addresses the issue of how the logical form relates to syntactic structure and shows how the feature system in the grammar can be used to identify the logical form in a rule-by-rule fashion. Chapter 10 discusses the important problem of ambiguity resolution and shows how semantic constraints or preferences can be used to identify the most plausible word senses and semantic structures. Chapter 11 discusses some alternate methods of semantic interpretation that have proven useful in existing systems and applications. Finally, Chapter 12 discusses a selection of more advanced issues in semantic interpretation, including the analysis of scoping dependencies, for the student with strong interests in semantic issues.

Semantics and Logical Form

- 8.1 Semantics and Logical Form
- 8.2 Word Senses and Ambiguity
- 8.3 The Basic Logical Form Language
- 8.4 Encoding Ambiguity in the Logical Form
- 8.5 Verbs and States in Logical Form
- 8.6 Thematic Roles
- 8.7 Speech Acts and Embedded Sentences

8.1 Semantics and Logical Form

8.8 Defining Semantic Structure: Model Theory

>> [back](#)

Allen 1995 : Natural Language Understanding

	<u>Contents</u>	<u>Preface</u>	<u>Introduction</u>	
<u>previous chapter</u>	<u>Part I Syntactic Processing</u>	<u>Part II Semantic Interpretation</u>	<u>Part III - Context / World Knowledge</u>	<u>next chapter</u>
	<u>Appendices</u>	<u>Bibliography</u>		
	<u>Summaries</u>	<u>Further Readings</u>	<u>Exercises</u>	

Chapter 2: Linguistic Background: An Outline of English Syntax

[2.1 Words](#)

[2.2 The Elements of Simple Noun Phrases](#)

[2.3 Verb Phrases and Simple Sentences](#)

[2.4 Noun Phrases Revisited](#)

[2.5 Adjective Phrases](#)

[2.6 Adverbial Phrases](#)

[Summary](#)

[Related Work and Further Readings](#)

[2.1 Words](#)

[Exercises for Chapter 2](#)

[Allen 1995: Linguistic Background: An Outline of English Syntax 23]

This chapter provides background material on the basic structure of English syntax for those who have not taken any linguistics courses. It reviews the major phrase categories and identifies their most important subparts. Along the way all the basic word categories used in the book are introduced. While the only structures discussed are those for English, much of what is said applies to nearly all other European languages as well. The reader who has some background in linguistics can quickly skim this chapter, as it does not address any computational issues. You will probably want to use this chapter as a reference source as you work through the rest of the chapters in Part I.

Section 2.1 describes issues related to words and word classes. Section 2.2 describes simple noun phrases, which are then used in Section 2.3 to describe simple verb phrases. Section 2.4 considers complex noun phrases that

include embedded sentences such as relative clauses. The remaining sections briefly cover other types of phrases: adjective phrases in Section 2.5 and adverbial phrases in Section 2.6.

2.1 Words

At first glance the most basic unit of linguistic structure appears to be the word. The word, though, is far from the fundamental element of study in linguistics; it is already the result of a complex set of more primitive parts. The study of morphology concerns the construction of words from more basic components corresponding roughly to meaning units. There are two basic ways that new words are formed, traditionally classified as **inflectional** forms and **derivational** forms. Inflectional forms use a root form of a word and typically add a **suffix** so that the word appears in the appropriate form given the sentence. Verbs are the best examples of this in English. Each verb has a basic form that then is typically changed depending on the subject and the tense of the sentence. For example, the verb *sigh* will take suffixes such as *-s*, *-ing*, and *-ed* to create the verb forms *sighs*, *sighing*, and *sighed*, respectively. These new words are all verbs and share the same basic meaning. Derivational morphology involves the derivation of new words from other forms. The new words may be in completely different categories from their subparts. For example, the noun *friend* is made into the adjective *friendly* by adding the suffix *-ly*. A more complex derivation would allow you to derive the noun *friendliness* from the adjective form. There are many interesting issues concerned with how words are derived and how the choice of word form is affected by the syntactic structure of the sentence that constrains it.

Traditionally, linguists classify words into different categories based on their uses. Two related areas of evidence are used to divide words into categories. The first area concerns the word's contribution to the meaning of the phrase that contains it, and the second area concerns the actual syntactic structures in which the word may play a

role. For example, you might posit the class noun as those words that can be used to identify the basic type (If object, concept, or place being discussed, and adjective as those words that further qualify the object)

[Allen 1995: Linguistic Background: An Outline of English Syntax 24]

concept, or place. Thus *green* would be an adjective and *book* a noun, as shown in the phrases *the green book* and *green books*. But things are not so simple:

green might play the role of a noun, as in *That green is lighter than the other*, and *book* might play the role of a modifier, as in *the book worm*. In fact, most nouns seem to be able to be used as a modifier in some situations. Perhaps the classes should be combined, since they overlap a great deal. But other forms of evidence exist. Consider what words could complete the sentence *it's so...* You might say *it's so green*, *it's so hot*, *it's so true*, and so on. Note that although *book* can be a modifier in *the book worm*, you cannot say **it's so book* about anything. Thus there are two classes of modifiers: adjective modifiers and noun modifiers.

Consider again the case where adjectives can be used as nouns, as in *the green*. Not all adjectives can be used in such a way. For example, the noun phrase *the hot* can be used, given a context where there are hot and cold plates, in a sentence such as *The hot are on the table*. But this refers to the hot plates; it cannot refer to hotness in the way the phrase *the green* refers to green. With this evidence you could subdivide adjectives into two subclasses—those that can also be used to describe a concept or quality directly, and those that cannot. Alternatively, however, you

can simply say that *green* is ambiguous between being an adjective or a noun and, therefore, falls in both classes. Since *green* can behave like any other noun, the second solution seems the most direct.

Using similar arguments, we can identify four main classes of words in English that contribute to the meaning of sentences. These classes are nouns, adjectives, verbs, and adverbs. Sentences are built out of phrases centered on these four word classes. Of course, there are many other classes of words that are necessary to form sentences, such as articles, pronouns, prepositions, particles, quantifiers, conjunctions, and so on. But these classes are fixed in the sense that new words in these classes are rarely introduced into the language. New nouns, verbs, adjectives and adverbs, on the other hand, are regularly introduced into the language as it evolves. As a result, these classes are called the open class words, and the others are called the closed class words.

A word in any of the four open classes may be used to form the basis of a phrase. This word is called the head of the phrase and indicates the type of thing, activity, or quality that the phrase describes. For example, with noun phrases, the head word indicates the general classes of objects being described. The phrases

the dog

the mangy dog

the mangy dog at the pound

are all noun phrases that describe an object in the class of dogs. The first describes a member from the class of all dogs, the second an object from the class of mangy dogs, and the third an object from the class of mangy dogs that are at the pound. The word *dog* is the head of each of these phrases.

[Allen 1995: Linguistic Background: An Outline of English Syntax 25]

Noun Phrases	Verb Phrases
The president <i>of the company</i>	<i>looked up the chimney</i>
His desire <i>to succeed</i>	<i>believed that the world was flat</i>
Several challenges <i>from the opposing team</i>	<i>ate the pizza</i>

Adjective Phrases	Adverbial Phrases
-------------------	-------------------

Adjective Phrases	Adverbial Phrases
<i>easy to assemble</i>	<i>rapidly like a bat</i>
<i>happy that he'd won the prize</i>	<i>intermittently throughout the day</i>
<i>angry as a hippo</i>	<i>inside the house</i>

Figure 2.1 Examples of heads and complements

Similarly, the adjective phrases

hungry

very hungry

hungry as a horse

all describe the quality of hunger. In each case the word *hungry* is the head.

In some cases a phrase may consist of a single head. For example, the word *sand* can be a noun phrase, *hungry* can be an adjective phrase, and *walked* can be a verb phrase. In many other cases the head requires additional phrases following it to express the desired meaning. For example, the verb *put* cannot form a verb phrase in isolation; thus the following words do not form a meaningful sentence:

**Jack put.*

To be meaningful, the verb *put* must be followed by a noun phrase and a phrase describing a location, as in the verb phrase *put the dog in the house*. The phrase or set of phrases needed to complete the meaning of such a head is called the complement of the head. In the preceding phrase *put* is the head and *the dog in the house* is the complement. Heads of all the major classes may require complements. Figure 2.1 gives some examples of phrases, with the head indicated by boldface and the complements by italics, in the remainder of this chapter, we will look at these different types of phrases in more detail and see how they are structured and how they contribute to the meaning of sentences.

>> [back](#)

2.2 The Elements of Simple Noun Phrases

Noun phrases (NPs) are used to refer to things: objects, places, concepts, events, qualities, and so on. The simplest NP consists of a single pronoun: *he*, *she*, *they*, *you*, *me*, *it*, *i*, and so on. Pronouns can refer to physical objects, as in the sentence

It hid under the rug.

[Allen 1995: Linguistic Background: An Outline of English Syntax 26]

to events, as in the sentence

Once I opened the door, I regretted it for months.

and to qualities, as in the sentence

He was so angry, but he didn't show it.

Pronouns do not take any modifiers except in rare forms, as in the sentence "He who hesitates is lost".

Another basic form of noun phrase consists of a name or proper noun, such as *John* or *Rochester*. These nouns appear in capitalized form in carefully written English. Names may also consist of multiple words, as in the *New York Times* and *Stratford-on-Avon*.

Excluding pronouns and proper names, the head of a noun phrase is usually a common noun. Nouns divide into two main classes:

count nouns — nouns that describe specific objects or sets of objects.

mass nouns — nouns that describe composites or substances.

Count nouns acquired their name because they can be counted. There may be one *dog* or many *dogs*, one *book* or several *books*, one *crowd* or several *crowds*. If a single count noun is used to describe a whole class of objects, it must be in its plural form. Thus you can say *Dogs are friendly* but not **Dog is friendly*.

Mass nouns cannot be counted. There may be *some water*, *some wheat*, or *some sand*. If you try to count with a mass noun, you change the meaning. For example, *some wheat* refers to a portion of some quantity of wheat, whereas *one wheat* is a single type of wheat rather than a single grain of wheat. A mass noun can be used to describe a whole class of material without using a plural form. Thus you say *Water is necessary for life*, not **Waters are necessary for life*.

In addition to a head, a noun phrase may contain specifiers and qualifiers preceding the head. The qualifiers further describe the general class of objects identified by the head, while the specifiers indicate how many such objects are being described, as well as how the objects being described relate to the speaker and hearer. Specifiers are constructed out of ordinals (such as *first* and *second*), cardinals (such as *one* and *two*), and determiners. Determiners can be subdivided into the following general classes:

articles — the words *the*, *a*, and *an*.

demonstratives — words such as *this*, *that*, *these*, and *those*.

possessives — noun phrases followed by the suffix ‘s, such as *John’s* and *the fat man’s*, as well as possessive pronouns, such as *her*, *my*, and *whose*.

wh-determiners — words used in questions, such as *which* and *what*.

quantifying determiners — words such as *some*, *every*, *most*, *no*, *any*, *both*, and *half*

[Allen 1995: Linguistic Background: An Outline of English Syntax 27]

Number	First Person	Second Person	Third Person
			he (masculine)
singular	I	you	she (feminine)
			it (neuter)
plural	we	you	they

Figure 2.2 Pronoun system (as subject)

Number	First Person	Second Person	Third Person
singular	my	your	his, her, its
plural	our	your	their

Figure 2.3 Pronoun system (possessives)

A simple noun phrase may have at most one determiner, one ordinal, and one cardinal. It is possible to have all three, as in *the first three contestants*. An exception to this rule exists with a few quantifying determiners such as *many*, *few*, *several*, and *little*. These words can be preceded by an article, yielding noun phrases such as *the few songs we knew*. Using this evidence, you could subcategorize the quantifying determiners into those that allow this and those that don't, but the present coarse categorization is fine for our purposes at this time.

The qualifiers in a noun phrase occur after the specifiers (if any) and before the head. They consist of adjectives and nouns being used as modifiers. The following are more precise definitions:

adjectives - words that attribute qualities to objects yet do not refer to the qualities themselves (for example, *angry* is an adjective that attributes the quality of anger to something).

noun modifiers - mass or count nouns used to modify another noun,

as in *the cook book* or *the ceiling paint can*.

Before moving on to other structures, consider the different inflectional forms that nouns take and how they are realized in English. Two forms of nouns - the singular and plural forms - have already been mentioned. Pronouns take forms based on person (first, second, and third) and gender (masculine, feminine, and neuter). Each of these distinctions reflects a systematic analysis that is almost wholly explicit in some languages, such as Latin, while implicit in others. In French, for example, nouns are classified by their gender. In English many of these distinctions are not explicitly marked except in a few cases. The pronouns provide the best example of this. They distinguish number, person, gender, and case (that is, whether they are used as possessive, subject, or object), as shown in Figures 2.2 through 2.4.

[Allen 1995: Linguistic Background: An Outline of English Syntax 28]

Number	First Person	Second Person	Third Person
--------	--------------	---------------	--------------

Number	First Person	Second Person	Third Person
singular	me	you	her / him / it
plural	us	you	them

Figure 2.4 Pronoun system (as object)

Mood	Example
declarative (or assertion)	The cat is sleeping. yes/no question Is the cat sleeping?
wh-question	What is sleeping? or Which cat is sleeping? imperative (or command) Shoot the cat

Figure 2.5 Basic moods of sentences

Form	Examples	Example Uses
base	hit, cry, go, be	Hit the ball!
		I want to go.
simple present	hit, cries, go, am	The dog cries every day.
		I am thirsty.
simple past	hit, cried, went, was	I was thirsty.
		I went to the store.
present participle	hitting, crying, going, being	I'm going to the store.

Form	Examples	Example Uses
		Being the last in line aggravates me.
past participle	hit, cried, gone, been	I've been there before. The cake was gone.

Figure 2.6 The five verb forms

>> [back](#)

2.3 Verb Phrases and Simple Sentences

While an NP is used to refer to things, a sentence (S) is used to assert, query, or command. You may assert that some sentence is true, ask whether a sentence is true, or command someone to do something described in the sentence. The way a sentence is used is called its mood. Figure 2.5 shows four basic sentence moods.

A simple declarative sentence consists of an NP, the subject, followed by a verb phrase (VP), the predicate. A simple VP may consist of some adverbial modifiers followed by the head verb and its complements. Every verb must appear in one of the five possible forms shown in Figure 2.6.

[Allen 1995: Linguistic Background: An Outline of English Syntax 29]

Tense	The Verb Sequence	Example

Tense	The Verb Sequence	Example
simple present	simple present	He walks to the store.
simple past	simple past	He walked to the store.
simple future	<i>will</i> + infinitive	He will walk to the store.
present perfect	<i>have</i> in present + past participle	He has walked to the store.
future perfect	<i>will</i> + <i>have</i> in infinitive + past participle	I will have walked to the store.
past perfect (or pluperfect)	<i>have</i> in past + past participle	I had walked to the store.

Figure 2.7 The basic tenses

Tense	Structure	Example
present progressive	<i>be</i> in present + present participle	He is walking.
past progressive	<i>be</i> in past + present participle	He was walking.
future progressive	<i>will + be</i> in infinitive + present participle	He will be walking.
present perfect progressive	<i>have</i> in present + <i>be</i> in past participle + present participle	He has been walking.

Tense	Structure	Example
future perfect progressive	<i>will + have</i> in present + <i>be</i> as past participle + present participle	He will have been walking.
past perfect progressive	<i>have</i> in past + <i>be</i> in past participle + present participle	He had been walking.

Figure 2.8 The progressive tenses

Verbs can be divided into several different classes: the auxiliary verbs, such as *be*, *do*, and *have*; the modal verbs, such as *will*, *can*, and *could*; and the main verbs, such as *eat*, *ran*, and *believe*. The auxiliary and modal verbs

usually take a verb phrase as a complement, which produces a sequence of verbs, each the head of its own verb phrase. These sequences are used to form sentences with different tenses.

The tense system identifies when the proposition described in the sentence is said to be true. The tense system is complex; only the basic forms are outlined in Figure 2.7. In addition, verbs may be in the progressive tense. Corresponding to the tenses listed in Figure 2.7 are the progressive tenses shown in Figure 2.8.

[Allen 1995: Linguistic Background: An Outline of English Syntax 30]

	First	Second	Third
Singular	I am	you are	he is
	I walk	you walk	she walks
Plural	we are	you are	they are
	we walk	you walk	they walk

Figure 2.9 Person/number forms of verbs

Each progressive tense is formed by the normal tense construction of the verb *be* followed by a present participle.

Verb groups also encode person and number information in the first word in the verb group. The person and number must agree with the noun phrase that is the subject of the verb phrase. Some verbs distinguish nearly all the possibilities, but most verbs distinguish only the third person singular (by adding an *-s* suffix). Some examples are shown in Figure 2.9.

Transitivity and Passives

The last verb in a verb sequence is called the main verb, and is drawn from the open class of verbs. Depending on the verb, a wide variety of complement structures are allowed. For example, certain verbs may stand alone with no complement. These are called intransitive verbs and include examples such as *laugh* (for example, *Jack laughed*) and *run* (for example, *He will have been running*). Another common complement form requires a noun phrase to follow the verb. These are called transitive verbs and include verbs such as *find* (for example, *Jack*

found a key). Notice that *find* cannot be intransitive (for example, **Jack found* is not a reasonable sentence), whereas *laugh* cannot be transitive (for example, **Jack laughed a key* is not a reasonable sentence). A verb like *run*, on the other hand, can be transitive or intransitive, but the meaning of the verb is different in each case (for example, *Jack ran* vs. *Jack ran the machine*).

Transitive verbs allow another form of verb group called the passive form, which is constructed using a *be* auxiliary followed by the past participle. In the passive form the noun phrase that would usually be in the object position is used in the subject position, as can be seen by the examples in Figure 2.10. Note that tense is still carried by the initial verb in the verb group. Also, even though the first noun phrase semantically seems to be the object of the verb in passive sentences, it is syntactically the subject. This can be seen by checking the pronoun forms. For example, *I was hit* is correct, not **Me was hit*. Furthermore, the tense and number agreement is between the verb and the syntactic subject. Thus you say *I was hit by them*, not **I were hit by them*.

Some verbs allow two noun phrases to follow them in a sentence; for example, *Jack gave Sue a book* or *Jack found me a key*. In such sentences the

[Allen 1995: Linguistic Background: An Outline of English Syntax 31]

Active Sentence	Related Passive Sentence
-----------------	--------------------------

Active Sentence	Related Passive Sentence
Jack saw the ball.	The ball was seen by Jack.
I will find the clue.	The clue will be found by me.
Jack hit me.	I was hit by Jack.

Figure 2.10 Active sentences with corresponding passive sentences

second NP corresponds to the object NP outlined earlier and is sometimes called the direct object. The other NP is called the indirect object. Generally, such sentences have an equivalent sentence where the indirect object appears with a preposition, as in *Jack gave a book to Sue* or *Jack found a key for me*.

Particles

Some verb forms are constructed from a verb and an additional word called a particle. Particles generally overlap with the class of prepositions considered in the next section. Some examples are *up*, *out*, *over*, and *in*. With verbs such as *look*, *take*, or *put*, you can construct many different verbs by combining the verb with a particle (for example, *look up*, *look out*, *look over*, and so on). In some sentences the difference between a particle and a preposition results in two different readings for the same sentence. For example, *look over the paper* would mean reading the paper, if you consider *over* a particle (the verb is *look over*). In contrast, the same sentence would mean looking at something else behind or above the paper, if you consider *over* a preposition (the verb is *look*).

You can make a sharp distinction between particles and prepositions when the object of the verb is a pronoun. With a verb-particle sentence, the pronoun must precede the particle, as in *I looked it up*. With the prepositional reading, the pronoun follows the preposition, as in *I looked up it*. Particles also may follow the object NP. Thus you can say *I gave up the game to Mary* or *I gave the game up to Mary*. This is not allowed with prepositions; for example, you cannot say **I climbed the ladder up*.

Clausal Complements

Many verbs allow clauses as complements. Clauses share most of the same properties of sentences and may have a subject, indicate tense, and occur in passivized forms. One common clause form consists of a sentence form preceded by the complementizer *that*, as in *that Jack ate the pizza*. This clause will be identified by the expression

S[that], indicating a special subclass of S structures. This clause may appear as the complement of the verb *know*, as in *Sam knows that Jack ate the pizza*. The passive is possible, as in *Sam knows that the pizza was eaten by Jack*.

[Allen 1995: Linguistic Background: An Outline of English Syntax 32]

Another clause type involves the infinitive form of the verb. The VP[inf] clause is simply a VP starting in the infinitive form, as in the complement of the verb *wish* in *Jack wishes to eat the pizza*. An infinitive sentence S[inf] form is also possible where the subject is indicated by a *for* phrase, as in *Jack wishes for Sam to eat the pizza*.

Another important class of clauses are sentences with complementizers that are wh-words, such as *who*, *what*, *where*, *why*, *whether*, and *how many*. These question clauses, S[WH], can be used as a complement of verbs such as *know*, as in *Sam knows whether we went to the party* and *The police know who committed the crime*.

Prepositional Phrase Complements

Many verbs require complements that involve a specific prepositional phrase (PP). The verb *give* takes a complement consisting of an NP and a PP with the preposition *to*, as in *Jack gave the book to the library*. No other preposition can be used. Consider

**Jack gave the book from the library.* (OK only if *from the library* modifies book.)

In contrast, a verb like *put* can take any PP that describes a location, as in

Jack put the book in the box.

Jack put the book inside the box.

Jack put the book by the door.

To account for this, we allow complement specifications that indicate prepositional phrases with particular prepositions. Thus the verb *give* would have a complement of the form NP+PP[to]. Similarly the verb *decide* would have a complement form NP+PP[about], and the verb *blame* would have a complement form NP+PP[on], as in *Jack blamed the accident on the police*.

Verbs such as *put*, which take any phrase that can describe a location (complement NP+Location), are also common in English. While locations are typically prepositional phrases, they also can be noun phrases, such as *home*, or particles, such as *back* or *here*. A distinction can be made between phrases that describe locations and phrases that describe a path of motion, although many location phrases can be interpreted either way. The distinction can be made in some cases, though. For instance, prepositional phrases beginning with "to" generally indicate a path of motion. Thus they cannot be used with a verb such as "*put*" that requires a location (for example, **I put the ball to the box*). This distinction will be explored further in Chapter 4.

Figure 2.11 summarizes many of the verb complement structures found in English. A full list would contain over 40 different forms. Note that while the examples typically use a different verb for each form, most verbs will allow several different complement structures.

[Allen 1995: Linguistic Background: An Outline of English Syntax 33]

Verb	Complement Structure	Example
laugh	Empty (intransitive)	Jack laughed.
find	NP (transitive)	Jack found a key.
give	NP+NP (bitransitive)	Jack gave Sue the paper.
give	NP+PP[to]	Jack gave the book to the library.
reside	Location phrase	Jack resides in Rochester

Verb	Complement Structure	Example
put	NP+Location phrase	Jack put the book inside.
speak	PP[with]+PP[about]	Jack spoke with Sue about the book.
try	VP[to]	Jack tried to apologize.
tell	NP+VP[to]	Jack told the man to go.
wish	S[to]	Jack wished for the man to go.
keep	VP[ing]	Jack keeps hoping for the best.
catch	NP+VP[ing]	Jack caught Sam looking in his desk.
watch	NP+VP[base]	Jack watched Sam eat the pizza.
regret	S[that]	Jack regretted that he'd eaten the whole thing.
tell	NP+S[that]	Jack told Sue that he was sorry.

Verb	Complement Structure	Example
seem	ADJP	Jack seems unhappy in his new job.
think	NP+ADJP	Jack thinks Sue is happy in her job.
know	S[WH]	Jack knows where the money is.

Figure 2.11 Some common verb complement structures in English

>> [back](#)

2.4 Noun Phrases Revisited

Section 2.2 introduced simple noun phrases. This section considers more complex forms in which NPs contain sentences or verb phrases as subcomponents.

All the examples in Section 2.2 had heads that took the null complement. Many nouns, however, may take complements. Many of these fall into the class of complements that require a specific prepositional phrase. For example, the noun *love* has a complement form PP[of], as in *their love of France*, the noun *reliance* has the complement form PP[on], as in *his reliance on handouts*, and the noun *familiarity* has the complement form PP[with], as in *a familiarity with computers*.

Many nouns, such as *desire*, *reluctance*, and *research*, take an infinitive VP form as a complement, as in the noun phrases *his desire to release the guinea pig*, *a reluctance to open the case again*, and *the doctor's research to find a cure for cancer*. These nouns, in fact, can also take the S[infl] form, as in *my hope for John to open the case again*.

Noun phrases can also be built out of clauses, which were introduced in the last section as the complements for verbs. For example, a *that* clause (S_ithat_i) can be used as the subject of a sentence, as in the sentence *That George had the ring was surprising*. Infinitive forms of verb phrases (VPfintl) and sentences (S(in Ii)) can also function as noun phrases, as in the sentences "*To own a car would be*

[Allen 1995: Linguistic Background: An Outline of English Syntax 34]

delightful" and "*For us to complete a project on time would be unprecedented*". In addition, the gerundive forms (VP[ing] and S[ing]) can also function as noun phrases, as in the sentences *Giving up the game was unfortunate* and *John's giving up the game caused a riot*.

Relative clauses involve sentence forms used as modifiers in noun phrases. These clauses are often introduced by relative pronouns such as *who*, *which*, *that*, and so on, as in

The man who gave Bill the money...

The rug that George gave to Ernest...

The man whom George gave the money to ...

In each of these relative clauses, the embedded sentence is the same structure as a regular sentence except that one noun phrase is missing. If this missing NP is filled in with the NP that the sentence modifies, the result is a complete, sentence that captures the same meaning as what was conveyed by the relative clause. The missing NPs

in the preceding three sentences occur in the subject position, in the object position, and as object to a preposition, respectively. Deleting the relative pronoun and filling in the missing NP in each produces the following:

The man gave Bill the money.

George gave the rug to Ernest.

George gave the money to the man.

As was true earlier, relative clauses can be modified in the same ways as regular sentences. In particular, passive forms of the preceding sentences would be as follows:

Bill was given the money by the man.

The rug was given to Ernest by George.

The money was given to the man by George.

Correspondingly, these sentences could have relative clauses in the passive form as follows:

The man Bill was given the money by...

The rug that was given to Ernest by George...

The man whom the money was given to by George...

Notice that some relative clauses need not be introduced by a relative pronoun. Often the relative pronoun can be omitted, producing what is called a base relative clause, as in the NP *the man George gave the money to*. Yet another form deletes the relative pronoun and an auxiliary *be* form, creating a reduced relative clause, as in the NP *the man given the money*, which means the same as the NP *the man who was given the money*.

>> [back](#)

[Allen 1995: Linguistic Background: An Outline of English Syntax 35]

2.5 Adjective Phrases

You have already seen simple adjective phrases (ADJPs) consisting of a single adjective in several examples. More complex adjective phrases are also possible, as adjectives may take many of the same complement forms that occur with verbs. This includes specific prepositional phrases, as with the adjective *pleased*, which takes the complement form PP[with] (for example, *Jack was pleased with the prize*), or *angry* with the complement form PP[atl (for example, *Jack was angry at the committee*). *Angry* also may take an S[that] complement form, as in *Jack was angry that he was left behind*. Other adjectives take infinitive forms, such as the adjective *willing* with the complement form VP[inf], as in *Jack seemed willing to lead the chorus*.

These more complex adjective phrases are most commonly found as the complements of verbs such as *be* or *seem*, or following the head in a noun phrase. They generally cannot be used as modifiers preceding the heads of noun phrases (for example, consider **the angry at the committee man* vs. *the angry man* vs. *the man angry at the committee*).

Adjective phrases may also take a degree modifier preceding the head, as in the adjective phrase *very angry* or *somewhat fond of Mary*. More complex degree modifications are possible, as in *far too heavy* and *much more desperate*. Finally, certain constructs have degree modifiers that involve their own complement forms, as in *too stupid to come in out of the rain*, *so boring that everyone fell asleep*, and *as slow as a dead horse*.

>> [back](#)

2.6 Adverbial Phrases

You have already seen adverbs in use in several constructs, such as indicators of degree (for example, *very*, *rather*, *too*), and in location phrases (for example, *here*, *everywhere*). Other forms of adverbs indicate the manner in which something is done (for example, *slowly*, *hesitantly*), the time of something (for example, *now*, *yesterday*), or the frequency of something (for example, *frequently*, *rarely*, *never*).

Adverbs may occur in several different positions in sentences: in the sentence initial position (for example, *Then, Jack will open the drawer*), in the verb sequence (for example, *Jack then will open the drawer*, *Jack will then open the drawer*), and in the sentence final position (for example, *Jack opened the drawer then*). The exact restrictions on what adverb can go where, however, is quite idiosyncratic to the particular adverb.

In addition to these adverbs, adverbial modifiers can be constructed out of a wide range of constructs, such as prepositional phrases indicating, among other things, location (for example, *in the box*) or manner (for example, *in great haste*); noun phrases indicating, among other things, frequency (for example, *every day*):

or clauses indicating, among other things, the time (for example, *when the bomb exploded*). Such adverbial phrases, however, usually cannot occur except in the sentence initial or sentence final position. For example, we can say *Every day*

[Allen 1995: Linguistic Background: An Outline of English Syntax 36]

John opens his drawer or *John opens his drawer every day*, but not **John every day opens his drawer*.

Because of the wide range of forms, it generally is more useful to consider adverbial phrases (AD VPs) by function rather than syntactic form. Thus we can consider manner, temporal, duration, location, degree, and frequency adverbial phrases each as its own form. We considered the location and degree forms earlier, so here we will consider some of the others.

Temporal adverbials occur in a wide range of forms: adverbial particles (for example, *now*), noun phrases (for example, *today*, *yesterday*), prepositional phrases (for example, *at noon*, *during the fight*), and clauses (for example, *when the clock struck noon*, *before the fight started*).

Frequency adverbials also can occur in a wide range of forms: particles (for example, *often*), noun phrases (for example, *every day*), prepositional phrases (for example, *at every party*), and clauses (for example, *every time that John comes for a visit*).

Duration adverbials appear most commonly as prepositional phrases (for example, *for three hours*, *about 20 feet*) and clauses (for example, *until the moon turns blue*).

Manner adverbials occur in a wide range of forms, including particles (for example, *slowly*), noun phrases (for example, *this way*), prepositional phrases (for example, *in great haste*), and clauses (for example, *by holding the embers at the end of a stick*).

in the analyses that follow, adverbials will most commonly occur as modifiers of the action or state described in a sentence. As such, an issue arises as to how to distinguish verb complements from adverbials. One distinction is that adverbial phrases are always optional. Thus you should be able to delete the adverbial and still have a sentence with approximately the same meaning (missing, obviously, the contribution of the adverbial). Consider the sentences

Jack put the box by the door.

Jack ate the pizza by the door.

In the first sentence the prepositional phrase is clearly a complement, since deleting it to produce **Jack put the box* results in a nonsensical utterance. On the other hand, deleting the phrase from the second sentence has only a minor effect:

Jack ate the pizza is just a less general assertion about the same situation described by *Jack ate the pizza by the door.*

>> [back](#)

Summary

The major phrase structures of English have been introduced—namely, noun phrases, sentences, prepositional phrases, adjective phrases, and adverbial phrases. These will serve as the building blocks for the syntactic structures introduced in the following chapters.

>> [back](#)

[Allen 1995: Linguistic Background: An Outline of English Syntax 37]

Related Work and Further Readings

An excellent overview of English syntax is found in Baker (1989). The most comprehensive sources are books that attempt to describe the entire structure of English, such as Huddleston (1988), Quirk et al. (1972), and Leech and Svartvik (1975).

>> [back](#)

Exercises for Chapter 2

(easy) The text described several different example tests for distinguishing word classes. For example, nouns can occur in sentences of the form *I saw the X*, whereas adjectives can occur in sentences of the form *it's so X*. Give some additional tests to distinguish these forms and to distinguish between count nouns and mass nouns. State whether each of the following words can be used as an adjective, count noun, or mass noun. If the word is ambiguous, give all its possible uses.

milk, house, liquid, green, group, concept, airborne

2. (easy) Identify every major phrase (noun, verb, adjective, or adverbial phrases) in the following sentences. For each, indicate the head of the phrase and any complements of the head. Be sure to distinguish between complements and optional modifiers.

The man played his fiddle in the street.

The people dissatisfied with the verdict left the courtroom.

3. (easy) A very useful test for determining the syntactic role of words and phrases is the conjunction test. Conjunctions, such as *and* and *or*, tend to join together two phrases of the same type. For instance, you can conjoin nouns, as in *the man and woman*; noun phrases, as in *the angry men and the large dogs*; and adjectives

phrases, as in *the cow, angry and confused, broke the gate*. In each of the following sentences, identify the type of phrase being conjoined, and underline each phrase.

He was tired and hungrier than a herd of elephants.

We have never walked home or to the store from here.

The dog returned quickly and dropped the stick.

4. (easy) Explain in detail, using the terminology of this chapter, why each of the following sentences is ill formed. In particular, state what rule (given in this chapter) has been violated.

a. *He barked the wrong tree up.*

b. *She turned waters into wine.*

c. *Don't take many all the cookies!*

d. *I feel floor today.*

e. *They all laughed the boy.*

[Allen 1995: Linguistic Background: An Outline of English Syntax 38]

5. (easy) Classify the following verbs as being intransitive, transitive, or bitransitive (that is, it takes a two-NP complement). If the verb can be used in more than one of these forms, give each possible classification. Give an example sentence for each form to demonstrate your analysis.

- a. cry
- b. sing
- c. donate
- d. put

6. (easy) Using the verb *to be*, give example sentences that use the six basic tenses listed in Figure 2.7 and the six progressive tenses listed in Figure 2.8.

7. (easy) Using the verb *donate*, give examples of the passive form for each of the six basic tenses listed in Figure 2.7.

8. (easy) Classify the following verbs by specifying what different complement structures they allow, using the forms defined in Figure 2.11.

give, know, assume, insert

Give an example of an additional complement structure that is allowed by one of these verbs but not listed in the figure.

9. (easy) Find five verbs not discussed in this chapter that take an indirect object and, for each one, give a paraphrase of the same sentence using a prepositional phrase instead of the indirect object. Try for as wide a range as possible. Can you find one that cannot be paraphrased using either the preposition *to* or *for*?

10. (medium) Wh-questions are questions that use a class of words that includes *what, where, who, when, whose, which, and how*. For each of these words, give the syntactic categories (for example, verb, noun, noun group, adjective, quantifier, prepositional phrase, and so on) in which the words can be used. Justify each classification with some examples that demonstrate it. Use both positive and negative arguments as necessary (such as "it is one of these because ...," or "it can't be one of these even though it looks like it might, because. .

>> [back](#)

Allen 1995: Natural Language Understanding

	<u>Contents</u>	<u>Preface</u>	<u>Introduction</u>	
<u>previous chapter</u>	<u>Part I Syntactic Processing</u>	<u>Part II Semantic Interpretation</u>	<u>Part III - Context / World Knowledge</u>	<u>next chapter</u>
	<u>Appendices</u>	<u>Bibliography</u>		
	<u>Summaries</u>	<u>Further Readings</u>	<u>Exercises</u>	

1. Introduction to Natural Language Understanding

	<u>1.1 The Study of Language</u>
	<u>1.2 Applications of Natural Language Understanding</u>
	<u>1.3 Evaluating Language Understanding Systems</u>
	<u>1.4 The Different Levels of Language Analysis</u>
	<u>1.5 Representations and Understanding</u>
	<u>1.6 The Organization of Natural Language Understanding Systems</u>
	<u>Summary</u>
	<u>Related Work and Further Readings</u>

[1.1 The Study of Language](#)

[Exercises for Chapter 1](#)

[Allen 1995 : Chapter 1 - Introduction / 1]

This chapter describes the field of natural language understanding and introduces some basic distinctions. Section 1.1 discusses how natural language understanding research fits into the study of language in general. Section 1.2 discusses some applications of natural language understanding systems and considers what it means for a system to understand language. Section 1.3 describes how you might evaluate whether a system understands language. Section 1.4 introduces a few basic distinctions that are made when studying language, and Section 1.5 discusses how computational systems often realize these distinctions. Finally, Section 1.6 discusses how natural language systems are generally organized, and introduces the particular organization assumed throughout this book.

1.1 The Study of Language

Language is one of the fundamental aspects of human behavior and is a crucial component of our lives. In written form it serves as a long-term record of knowledge from one generation to the next. In spoken form it serves as our primary means of coordinating our day-to-day behavior with others. This book describes research about how language comprehension and production work. The goal of this research is to create computational models of language in enough detail that you could write computer programs to perform various tasks involving natural language. The ultimate goal is to be able to specify models that approach human performance in the linguistic tasks of reading, writing, hearing, and speaking. This book, however, is not concerned with problems related to the specific medium used, whether handwriting, keyboard input, or speech. Rather, it is concerned with the processes of comprehending and using language once the words are recognized. Computational models are useful both for scientific purposes — for exploring the nature of linguistic communication — and for practical purposes — for enabling effective human-machine communication.

Language is studied in several different academic disciplines. Each discipline defines its own set of problems and has its own methods for addressing them. The linguist, for instance, studies the structure of language itself, considering questions such as why certain combinations of words form sentences but others do not, and why a sentence can have some meanings but not others. The psycholinguist, on the other hand, studies the processes of human language production and comprehension, considering questions such as how people identify the appropriate structure of a sentence and when they decide on the appropriate meaning for words. The philosopher considers how words can mean anything at all and how they identify objects in the world. Philosophers also

consider what it means to have beliefs, goals, and intentions, and how these cognitive capabilities relate to language. The goal of the computational linguist is to develop a computational theory of language, using the notions of algorithms and data structures from computer science. Of course, to build a computational model, you must take advantage of what is known from all the other disciplines. Figure 1.1 summarizes these different approaches to studying language.

[Allen 1995 : Chapter 1 - Introduction / 2]

Discipline	Typical Problems	Tools
Linguists	How do words form phrases and sentences? What constrains the possible meanings for a sentence?	Intuitions about well-formedness and meaning; mathematical models of structure (for example, formal language theory, model theoretic semantics)

Discipline	Typical Problems	Tools
Psycholinguists	How do people identify the structure of sentences? How are word meanings identified? When does understanding take place?	Experimental techniques based on measuring human performance; statistical analysis of observations
Philosophers	What is meaning, and how do words and sentences acquire it? How do words identify objects in the world?	Natural language argumentation using intuition about counter-examples; mathematical models (for example, logic and model theory)
Computational Linguists	How is the structure of sentences identified? How can knowledge and reasoning be modeled? How can language be used to accomplish specific tasks?	Algorithms, data structures; formal models of representation and reasoning; AI techniques (search and representation methods)

Figure 1.1 The major disciplines studying language

As previously mentioned, there are two motivations for developing computational models. The scientific motivation is to obtain a better understanding of how language works. It recognizes that any one of the other traditional disciplines does not have the tools to completely address the problem of how language comprehension and production work. Even if you combine all the approaches, a comprehensive theory would be too complex to be studied using traditional methods. But we may be able to realize such complex theories as computer programs and then test them by observing how well they perform. By seeing where they fail, we can incrementally improve them. Computational models may provide very specific predictions about human behavior that can then be explored by the psycholinguist. By continuing in this process, we may eventually acquire a deep understanding of how human language processing occurs. To realize such a dream will take the combined efforts of linguists, psycholinguists, philosophers, and computer scientists. This common goal has motivated a new area of interdisciplinary research often called cognitive science.

The practical, or technological, motivation is that natural language processing capabilities would revolutionize the way computers are used. Since most of human knowledge is recorded in linguistic form, computers that could understand natural language could access all this information. In addition, natural language interfaces to computers would allow complex systems to be accessible to

[Allen 1995 : Chapter 1 - Introduction / 3]

BOX 1.1 Boxes and Optional Sections

This book uses several techniques to allow you to identify what material is central and what is optional. In addition, optional material is sometimes classified as advanced, indicating that you may need additional background not covered in this book to fully appreciate the text. Boxes, like this one, always contain optional material, either providing more detail on a particular approach discussed in the main, text or discussing additional issues that are related to the text. Sections and subsections may be marked as optional by means of an open dot (o) before the heading. Optional sections provide more breadth and depth to chapters, but are not necessary for understanding material in later chapters. Depending on your interests and focus, you can choose among the optional sections to fill out the core material presented in the regular sections. In addition, there are dependencies between the chapters, so that entire chapters can be skipped if the material does not address your interests. The chapter dependencies are not marked explicitly in the text, but a chart of dependencies is given in the preface.

everyone. Such systems would be considerably more flexible and intelligent than is possible with current computer technology. For technological purposes it does not matter if the model used reflects the way humans process language. It only matters that it works.

This book takes a middle ground between the scientific and technological goals. On the one hand, this reflects a belief that natural language is so complex that an ad hoc approach without a well-specified underlying theory will not be successful. Thus the technological goal cannot be realized without using sophisticated underlying theories on the level of those being developed by linguists, psycholinguists, and philosophers. On the other hand, the present state of knowledge about natural language processing is so preliminary that attempting to build a cognitively correct model is not feasible. Rather, we are still attempting to construct any model that appears to work.

The goal of this book is to describe work that aims to produce linguistically motivated computational models of language understanding and production that can be shown to perform well in specific example domains. While the book focuses on computational aspects of language processing, considerable space is spent introducing the relevant background knowledge from the other disciplines that motivates and justifies the computational approaches taken. It assumes only a basic knowledge of programming, although the student with some background in linguistics, artificial intelligence (AI), and logic will appreciate additional subtleties in the development.

>> [back](#)

1.2 Applications of Natural Language Understanding

A good way to define natural language research is to consider the different applications that researchers work on. As you consider these examples. It will

[Allen 1995 : Chapter 1 - Introduction / 4]

also be a good opportunity to consider what it would mean to say that a computer system understands natural language. The applications can be divided into two major classes: text-based applications and dialogue-based applications.

Text-based applications involve the processing of written text, such as books, newspapers, reports, manuals, e-mail messages, and so on. These are all reading-based tasks. Text-based natural language research is ongoing in applications such as

- finding appropriate documents on certain topics from a database of texts (for example, finding relevant books in a library)
- extracting information from messages or articles on certain topics (for example, building a database of all stock transactions described in the news on a given day)
- translating documents from one language to another (for example, producing automobile repair manuals in many different languages)
- summarizing texts for certain purposes (for example, producing a 3-page summary of a 1000-page government report)

Not all systems that perform such tasks must be using natural language understanding techniques in the way we mean in this book. For example, consider the task of finding newspaper articles on a certain topic in a large database. Many, techniques have been developed that classify documents by the presence of certain keywords in the text. You can then retrieve articles on a certain topic by looking for articles that contain the keywords associated with that topic. Articles on law, for instance, might contain the words "*lawyer*", "*court*", "*sue*", "*affidavit*", and so on, while articles on stock transactions might contain words such as "*stocks*", "*takeover*", "*leveraged buyout*", "*options*", and so on. Such a system could retrieve articles on any topic that has been predefined by a set of keywords. Clearly, we would not say that this system is understanding the text; rather, it is using a simple matching technique. While such techniques may produce useful applications, they are inherently limited. It is very unlikely, for example, that they could be extended to handle complex retrieval tasks that are easily expressed in natural language, such as the query "*Find me all articles on leveraged buyouts involving more than 100 million dollars that were attempted but failed during 1986 and 1990*". To handle such queries, the system would have to be able to extract enough information from each article in the database to determine

whether the article meets the criteria defined by the query; that is, it would have to build a representation of the information in the articles and then use the representation to do the retrievals. This identifies a crucial characteristic of an understanding system: it must compute some representation of the information that can be used for later inference.

Consider another example. Some machine translation systems have been built that are based on pattern matching; that is, a sequence of words in one language is associated with a sequence of words in another language. The

[Allen 1995 : Chapter 1 - Introduction / 5]

translation is accomplished by finding the best set of patterns that match the input and producing the associated output in the other language. This technique can produce reasonable results in some cases but sometimes produces completely wrong translations because of its inability to use an understanding of content to disambiguate word senses and sentence meanings appropriately. In contrast, other machine translation systems operate by producing a representation of the meaning of each sentence in one language, and then producing a sentence in the other language that realizes the same meaning. This latter approach, because it involves the computation of a representation of meaning, is using natural language understanding techniques.

One very attractive domain for text-based research is story understanding. In this task the system processes a story and then must answer questions about it. This is similar to the type of reading comprehension tests used in schools

and provides a very rich method for evaluating the depth of understanding the system is able to achieve.

Dialogue-based applications involve human-machine communication. Most naturally this involves spoken language, but it also includes interaction using keyboards. Typical potential applications include

- question-answering systems, where natural language is used to query a database (for example, a query system to a personnel database)
- automated customer service over the telephone (for example, to perform banking transactions or order items from a catalogue)
- tutoring systems, where the machine interacts with a student (for example, an automated mathematics tutoring system)
- spoken language control of a machine (for example, voice control of a VCR or computer)
- general cooperative problem-solving systems (for example, a system that helps a person plan and schedule freight shipments)

Some of the problems faced by dialogue systems are quite different than in text-based systems. First, the language used is very different, and the system needs to participate actively in order to maintain a natural, smooth-flowing dialogue. Dialogue requires the use of acknowledgments to verify that things are understood, and an ability to

both recognize and generate clarification sub-dialogues when something is not clearly understood. Even with these differences, however, the basic processing techniques are fundamentally the same.

It is important to distinguish the problems of speech recognition from the problems of language understanding. A speech recognition system need not involve any language understanding. For instance, voice-controlled computers and VCRs are entering the market now. These do not involve natural language understanding in any general way. Rather, the words recognized are used as commands, much like the commands you send to a VCR using a remote control. Speech recognition is concerned only with identifying the words spoken from a

[Allen 1995 : Chapter 1 - Introduction / 6]

given speech signal, not with understanding how words are used to communicate. To be an understanding system, the speech recognizer would need to feed its input to a natural language understanding system, producing what is often called a spoken language understanding system.

With few exceptions, all the techniques discussed in this book are equally relevant for text-based and dialogue-based language understanding, and apply equally well whether the input is text, keyboard, or speech. The key characteristic of any understanding system is that it represents the meaning of sentences in some representation language that can be used later for further processing.

>> [back](#)

1.3 Evaluating Language Understanding Systems

As you can see, what counts as understanding might vary from application to application. If this is so, how can you tell if a system works? One obvious way to evaluate a system is to run the program and see how well it performs the task it was designed to do. If the program is meant to answer questions about a database of facts, you might ask it questions to see how good it is at producing the correct answers. If the system is designed to participate in simple conversations on a certain topic, you might try conversing with it. This is called black box evaluation because it evaluates system performance without looking inside to see how it works. While ultimately this method of evaluation may be the best test of a system's capabilities, it is problematic in the early stages of research because early evaluation results can be misleading. Sometimes the techniques that produce the best results in the short term will not lead to the best results in the long term. For instance, if the overall performance of all known systems in a given application is uniformly low, few conclusions can be drawn. The fact that one system was correct 50 percent of the time while another was correct only 40 percent of the time says nothing about the long-term viability of either approach. Only when the success rates become high, making a practical application feasible, can much significance be given to overall system performance measures.

An alternative method of evaluation is to identify various subcomponents of a system and then evaluate each one with appropriate tests. This is called glass box evaluation because you look inside at the structure of the system. The problem with glass box evaluation is that it requires some consensus on what the various components of a natural language system should be. Achieving such a consensus is an area of considerable activity at the present.

Despite the difficulties, evaluations are of crucial importance to the field, as very little can be concluded from examining a few examples that a particular system can handle. If you cannot interact freely with the program yourself but must depend on transcripts of others' interactions, you can easily be misled into overestimating a program's abilities. It is very hard for people not to attribute much more intelligence to the program than is actually present.

Consider a concrete example — the ELIZA program that was developed in the mid-1960s at MIT. This program must be one of the most popular AI

[Allen 1995 : Chapter 1 - Introduction / 7]

USER	ELIZA
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.

USER	ELIZA
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?

Figure 1.2 A dialogue with ELIZA

programs at all time, and versions of it exist for most machines, including most personal computers. ELIZA was never claimed to embody a theory of language comprehension and production, but it serves as an excellent example as its behavior initially seems impressive. The system plays the role of a therapist and, to obtain the best results, the user should correspondingly play the role of a patient. Figure 1.2 presents a transcript of this system in operation. Given this transcript, or even playing with the system yourself for a few minutes, ELIZA's performance certainly seems impressive.

Here is a simple description of how ELIZA works. There is a database of particular words that are called keywords. For each keyword, the system stores an integer, a pattern to match against the input, and a specification of the output. The algorithm is as follows: Given a sentence S, find a keyword in S whose pattern matches S. If there is more than one keyword, pick the one with the highest integer value. Use the output specification that is associated with this keyword to generate the next sentence. If there are no keywords, generate an innocuous continuation statement, such as "*Tell me more*" or "*Go on*".

Figure 1.3 shows a fragment of a database of keywords. In this database a pattern consists of words and variables. The prefix ? before a letter indicates a variable, which can match any sequence of words. For example, the pattern

?X are you ?Y

would match the sentence "*Why are you looking at me?*", where the variable ?X matches "*Why*" and "?Y" matches "*looking at me*". The output specification may also use the same variables. In this case, ELIZA inserts the words that match the variables in the input into the output after making some minor changes in the

[Allen 1995 : Chapter 1 - Introduction / 8]

Word	Rank	Pattern	Outputs
alike	10	?X	In what way? What resemblance do you see?
are	3	?X are you ?Y	Would you prefer it if I weren't ?Y?

Word	Rank	Pattern	Outputs
	3	?X are ?Y	What if they were not ?Y?
always	5	?X	Can you think of a specific example? When? Really, always?
what	2	?X	Why do you ask? Does that interest you?

Figure 1.3 Sample data from ELIZA

pronouns (for example, replacing "me" with "you"). Thus, for the pattern above, if the output specification is

Would you prefer it if I weren't ?Y?

the rule would generate a response "*Would you prefer it if I weren't looking at you?*" When the database lists multiple output specifications for a given pattern, ELIZA selects a different one each time a keyword rule is used, thereby preventing unnatural repetition in the conversation. Using these rules, you can see how ELIZA produced the first two exchanges in the conversation in Figure 1.2. ELIZA generated the first response from the first output of the keyword "alike" and the second response from the first output of the keyword "always".

This description covers all of the essential points of the program. You will probably agree that the program does not understand the conversation it is participating in. Rather, it is a collection of tricks. Why then does ELIZA appear to function so well? There are several reasons. Perhaps the most important reason is that, when people hear or read a sequence of words that they understand as a sentence, they attribute meaning to the sentence and assume that the person (or machine) that produced the sentence actually intended that meaning. People are extremely good at distinguishing word meanings and interpreting sentences to fit the context. Thus ELIZA appears to be intelligent because you use your own intelligence to make sense of what it says.

Other crucial characteristics of the conversational setting also aid in sustaining the illusion of intelligence. For instance, the system does not need any world knowledge because it never has to make a claim, support an argument, or answer a question. Rather, it simply asks a series of questions. Except in a patient-therapist situation, this would be unacceptable. ELIZA evades all direct questions by responding with another question, such as "*Why do you ask?*" There is no way to force the program to say something concrete about any topic.

[Allen 1995 : Chapter 1 - Introduction / 9]

Even in such a restricted situation, however, it is relatively easy to demonstrate that the program does not understand. It sometimes produces completely off-the-wall responses. For instance, if you say "*Necessity is the mother of invention*", it might respond with "*Tell me more about your family*", based on its pattern for the word "*mother*". In addition, since ELIZA has no knowledge about the structure of language, it accepts gibberish just as readily as valid sentences. If you enter "*Green the adzabak are the a ran four*", ELIZA will respond with something like "*What if they were not the a ran four?*" Also, as a conversation progresses, it becomes obvious that the program does not retain any of the content in the conversation. It begins to ask questions that are inappropriate in light of earlier exchanges, and its responses in general begin to show a lack of focus. Of course, if you are not able to play with the program and must depend only on transcripts of conversations by others, you would have no way of detecting these flaws, unless they are explicitly mentioned.

Suppose you need to build a natural language program for a certain application in only six months. If you start to construct a general model of language understanding, it will not be completed in that time frame and so will perform miserably on the tests. An ELIZA-like system, however, could easily produce behavior like that previously discussed with less than a few months of programming and will appear to far outperform the other system in testing. The differences will be especially marked if the test data only includes typical domain interactions that are not designed to test the limits of the system. Thus, if we take short-term performance as our only criteria of progress, everyone will build and fine-tune ELIZA-style systems, and the field will not progress past the limitations of the simple approach.

To avoid this problem, either we have to accept certain theoretical assumptions about the architecture of natural language systems and develop specific evaluation measures for different components, or we have to discount overall evaluation results until some reasonably high level of performance is obtained. Only then will cross-system comparisons begin to reflect the potential for long-term success in the field.

>> [back](#)

1.4 The Different Levels of Language Analysis

A natural language-system must use considerable knowledge about the structure of the language itself, including what the words are, how words combine to form sentences, what the words mean, how word meanings contribute to sentence meanings, and so on. However, we cannot completely account for linguistic behavior without also taking into account another aspect of what makes humans intelligent — their general world knowledge and their reasoning abilities. For example, to answer questions or to participate in a conversation, a person not only must know a lot about the structure of the language being used, but also must know about the world in general and the conversational setting in particular.

[Allen 1995 : Chapter 1 - Introduction / 10]

The following are some of the different forms of knowledge relevant for natural language understanding:

Phonetic and phonological knowledge - concerns how words are related to the sounds that realize them. Such knowledge is crucial for speech-based systems and is discussed in more detail in Appendix C.

Morphological knowledge - concerns how words are constructed from more basic meaning units called morphemes. A morpheme is the primitive unit of meaning in a language (for example, the meaning of the word "*friendly*" is derivable from the meaning of the noun "*friend*" and the suffix "*-ly*", which transforms a noun into an adjective).

Syntactic knowledge - concerns how words can be put together to form correct sentences and determines what structural role each word plays in the sentence and what phrases are subparts of what other phrases.

Semantic knowledge - concerns what words mean and how these meanings combine in sentences to form sentence meanings. This is the study of context-independent meaning - the meaning a sentence has regardless of the context in which it is used.

Pragmatic knowledge - concerns how sentences are used in different situations and how use affects the interpretation of the sentence.

Discourse knowledge-concerns how the immediately preceding sentences affect the interpretation of the next sentence. This information is especially important for interpreting pronouns and for interpreting the temporal aspects of the information conveyed.

World knowledge - includes the general knowledge about the structure of the world that language users must have in order to, for example, maintain a conversation. It includes what each language user must know about the other user's beliefs and goals.

These definitions are imprecise and are more characteristics of knowledge than actual distinct classes of knowledge. Any particular fact might include aspects from several different levels, and an algorithm might need to draw from several different levels simultaneously. For teaching purposes, however, this book is organized into three parts, each describing a set of techniques that naturally cluster together. Part I focuses on syntactic and morphological processing, Part II focuses on semantic processing, and Part III focuses on contextual effects in general, including pragmatics, discourse, and world knowledge.

[Allen 1995 : Chapter 1 - Introduction / 11]

BOX 1.2 Syntax, Semantics, and Pragmatics

The following examples may help you understand the distinction between syntax, semantics, and pragmatics. Consider each example as a candidate for the initial sentence of this book, which you know discusses natural language processing:

1. Language is one of the fundamental aspects of human behavior and is a crucial component of our lives.
2. Green frogs have large noses.

3. Green ideas have large noses.
4. Large have green ideas nose.

Sentence 1 appears to be a reasonable start (I hope!). It agrees with all that is known about syntax, semantics, and pragmatics. Each of the other sentences violates one or more of these levels. Sentence 2 is well-formed syntactically and semantically, but not pragmatically. It fares poorly as the first sentence of the book because the reader would find no reason for using it. But however bad sentence 2 would be as a start, sentence 3 is much worse. Not only is it obviously pragmatically ill-formed, it is also semantically ill-formed. To see this, consider that you and I could argue about whether sentence 2 is true or not, but we cannot do so with sentence 3. I cannot affirm or deny sentence 3 in coherent conversation. However, the sentence does have some structure, for we can discuss what is wrong with it: Ideas cannot be green and, even if they could, they certainly cannot have large noses. Sentence 4 is even worse. In fact, it is unintelligible, even though it contains the same words as sentence 3. It does not even have enough structure to allow you to say what is wrong with it. Thus it is syntactically ill-formed. Incidentally, there are cases in which a sentence may be pragmatically well-formed but not syntactically well-formed. For example, if I ask you where you are going and you reply "I go store", the response would be understandable even though it is syntactically ill-formed. Thus it is at least pragmatically well-formed and may even be semantically well-formed.

>> [back](#)

1.5 Representations and Understanding

As previously stated, a crucial component of understanding involves computing a representation of the meaning of sentences and texts. Without defining the notion of representation, however, this assertion has little content. For instance, why not simply use the sentence itself as a representation of its meaning? One reason is that most words have multiple meanings, which we will call senses. The word "*cook*", for example, has a sense as a verb and a sense as a noun; "*dish*" has multiple senses as a noun as well as a sense as a verb; and "*still*" has senses as a noun, verb, adjective, and adverb. This ambiguity would inhibit the system from making the appropriate inferences needed to model understanding. The disambiguation problem appears much easier than it actually is because people do not generally notice ambiguity. While a person does not seem to consider each of the possible

[Allen 1995 : Chapter 1 - Introduction / 12]

senses of a word when understanding a sentence, a program must explicitly consider them one by one.

To represent meaning, we must have a more precise language. The tools to do this come from mathematics and logic and involve the use of formally specified representation languages. Formal languages are specified from very simple building blocks. The most fundamental is the notion of an atomic symbol which is distinguishable from any other atomic symbol simply based on how it is written. Useful representation languages have the following two properties:

- The representation must be precise and unambiguous. You should be able to express every distinct reading of a sentence as a distinct formula in the representation.
- The representation should capture the intuitive structure of the natural language sentences that it represents. For example, sentences that appear to be structurally similar should have similar structural representations, and the meanings of two sentences that are paraphrases of each other should be closely related to each other.

Several different representations will be used that correspond to some of the levels of analysis discussed in the last section. In particular, we will develop formal languages for expressing syntactic structure, for context-independent word and sentence meanings, and for expressing general world knowledge.

Syntax: Representing Sentence Structure

The syntactic structure of a sentence indicates the way that words in the sentence are related to each other. This structure indicates how the words are grouped together into phrases, what words modify what other words, and what words are of central importance in the sentence. In addition, this structure may identify the types of relationships that exist between phrases and can store other information about the particular sentence structure that may be needed for later processing. For example, consider the following sentences:

1. *John sold the book to Mary.*
2. *The book was sold to Mary by John.*

These sentences share certain structural properties. In each, the noun phrases are "*John*", "*Mary*", and "*the book*", and the act described is some selling action. In other respects, these sentences are significantly different. For instance, even though both sentences are always either true or false in the exact same situations, you could only give sentence 1 as an answer to the question "*What did John do for Mary?*" Sentence 2 is a much better continuation of a sentence beginning with the phrase "*After it fell in the river*", as sentences 3 and 4 show. Following the standard convention in linguistics, this book will use an asterisk (*) before any example of an ill-formed or questionable sentence.

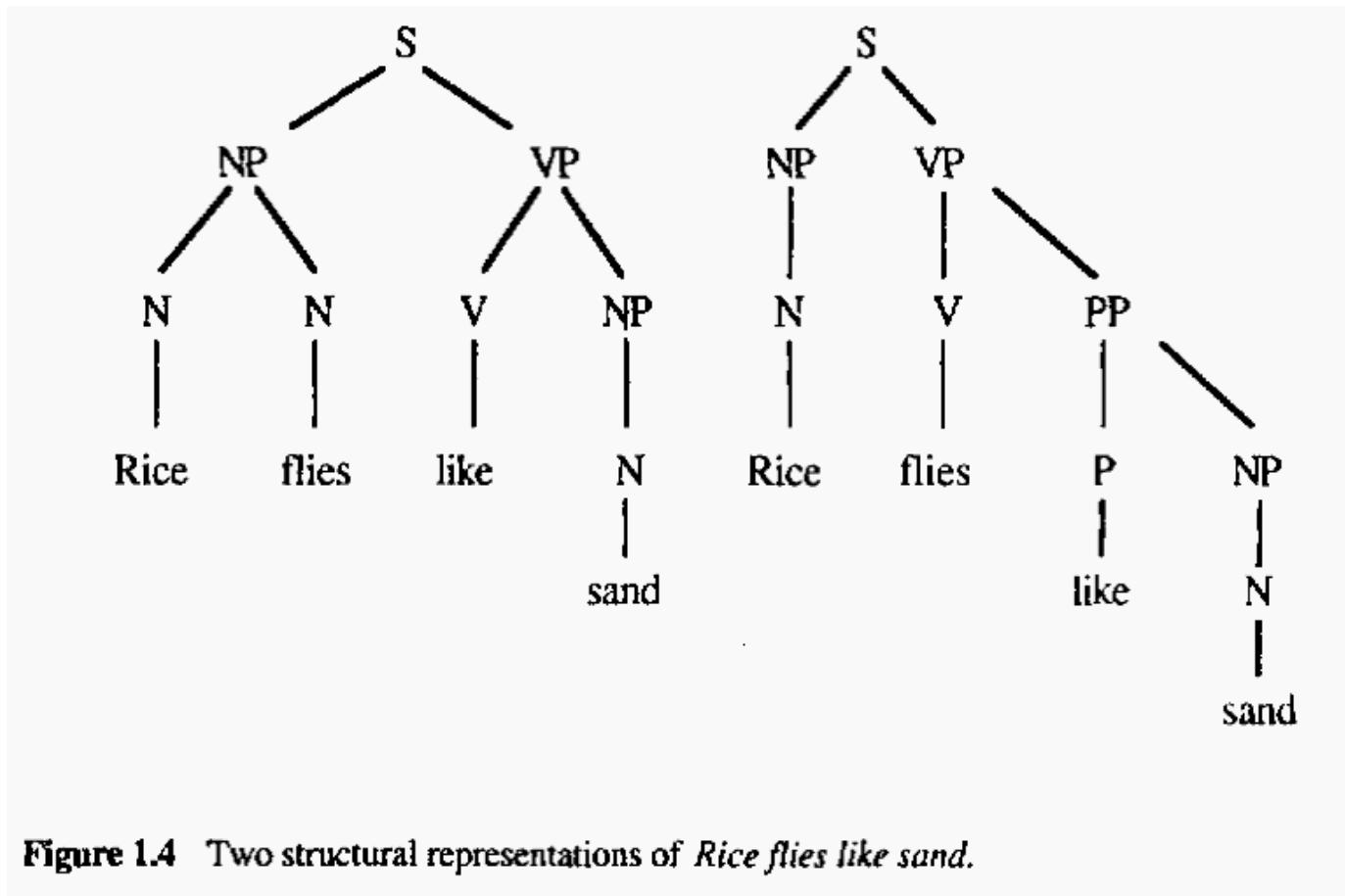


Figure 1.4 Two structural representations of *Rice flies like sand*.

Figure 1.4 Two structural representations of "*Rice flies like sand*".

3. *After it fell in the river, John sold Mary the book.
4. After it fell in the river, the book was sold to Mary by John.

Many other structural properties can be revealed by considering sentences that are not well-formed. Sentence 5 is ill-formed because the subject and the verb do not agree in number (the subject is singular and the verb is plural), while 6 is ill-formed because the verb *put* requires some modifier that describes where John put the object.

5. *John are in the corner.
6. *John put the book.

Making judgments on grammaticality is not a goal in natural language understanding. In fact, a robust system should be able to understand ill-formed sentences whenever possible. This might suggest that agreement checks

can be ignored, but this is not so. Agreement checks are essential for eliminating potential ambiguities. Consider sentences 7 and 8, which are identical except for the number feature of the main verb, yet represent two quite distinct interpretations.

7. *flying planes are dangerous.*

8. *flying planes is dangerous.*

If you did not check subject-verb agreement, these two sentences would be indistinguishable and ambiguous. You could find similar examples for every syntactic feature that this book introduces and uses.

Most syntactic representations of language are based on the notion of context-free grammars, which represent sentence structure in terms of what phrases are subparts of other phrases. This information is often presented in a tree form, such as the one shown in Figure 1.4, which shows two different structures

[Allen 1995 : Chapter 1 - Introduction / 14]

for the sentence "*Rice flies like sand*". In the first reading, the sentence is formed from a noun phrase (NP) describing a type of fly' rice flies, and a verb phrase (VP) that asserts that these flies like sand. In the second structure, the sentence is formed from a noun phrase describing a type of substance, rice, and a verb phrase stating that this substance flies like sand (say, if you throw it). The two structures also give further details on the structure of the noun phrase and verb phrase and identify the part of speech for each word. In particular, the word "*like*" is a verb (V) in the first reading and a preposition (P) in the second.

The Logical Form

The structure of a sentence doesn't reflect its meaning, however. For example, the NP "*the catch*" can have different meanings depending on whether the speaker is talking about a baseball game or a fishing expedition. Both these interpretations have the same syntactic structure, and the different meanings arise from an ambiguity concerning the sense of the word "*catch*". Once the correct sense is identified, say the fishing sense, there still is a problem in determining what fish are being referred to. The intended meaning of a sentence depends on the situation in which the sentence is produced. Rather than combining all these problems, this book will consider each one separately. The division is between context-independent meaning and context-dependent meaning. The fact that "*catch*" may refer to a baseball move or the results of a fishing expedition is knowledge about English and is independent of the situation in which the word is used. On the other hand, the fact that a particular noun

phrase "*the catch*" refers to what Jack caught when fishing yesterday is contextually dependent. The representation of the context-independent meaning of a sentence is called its logical form.

The logical form encodes possible word senses and identifies the semantic relationships between the words and phrases. Many of these relationships are often captured using an abstract set of semantic relationships between the verb and its NPs. In particular, in both sentences 1 and 2 previously given, the action described is a selling event, where "*John*" is the seller, "*the book*" is the object being sold, and "*Mary*" is the buyer. These roles are instances of the abstract semantic roles AGENT, THEME, and TO-POSS (for final possessor), respectively.

Once the semantic relationships are determined, some word senses may be impossible and thus eliminated from consideration. Consider the sentence

9. *Jack invited Mary to the Halloween ball.*

The word "*ball*", which by itself is ambiguous between the plaything that bounces and the formal dance event, can only take the latter sense in sentence 9, because the verb "*invite*" only makes sense with this interpretation. One of the key tasks in semantic interpretation is to consider what combinations of the individual word meanings can combine to create coherent sentence meanings. Exploiting such

[Allen 1995 : Chapter 1 - Introduction / 15]

interconnections between word meanings can greatly reduce the number of possible word senses for each word in a given sentence.

The Final Meaning Representation

The final representation needed is a general knowledge representation (KR), which the system uses to represent and reason about its application domain. This is the language in which all the specific knowledge based on the application is represented. The goal of contextual interpretation is to take a representation of the structure of a sentence and its logical form, and to map this into some expression in the KR that allows the system to perform the appropriate task in the domain. In a question-answering application, a question might map to a database query, in a story-understanding application, a sentence might map into a set of expressions that represent the situation that the sentence describes.

For the most part, we will assume that the first-order predicate calculus (FOPC) is the final representation language because it is relatively well known, well studied, and is precisely defined. While some inadequacies of FOPC will be examined later, these inadequacies are not relevant for most of the issues to be discussed.

>> [back](#)

1.6 The Organization of Natural Language Understanding Systems

This book is organized around the three levels of representation just discussed: syntactic structure, logical form, and the final meaning representation. Separating the problems in this way will allow you to study each problem in depth without worrying about other complications. Actual systems are usually organized slightly differently, however. In particular, Figure 1.5 shows the organization that this book assumes.

As you can see, there are interpretation processes that map from one representation to the other. For instance, the process that maps a sentence to its syntactic structure and logical form is called the parser. It uses knowledge about word and word meanings (the lexicon) and a set of rules defining the legal structures (the grammar) in order to assign a syntactic structure and a logical form to an input sentence. An alternative organization could perform syntactic processing first and then perform semantic interpretation on the resulting structures. Combining the two, however, has considerable advantages because it leads to a reduction in the number of possible interpretations, since every proposed interpretation must simultaneously be syntactically and semantically well formed. For example, consider the following two sentences:

10. Visiting relatives can be trying.

11. Visiting museums can be trying.

These two sentences have identical syntactic structure, so both are syntactically ambiguous. In sentence 10, the subject might be relatives who are visiting you or

[Allen 1995 : Chapter 1 - Introduction / 16]

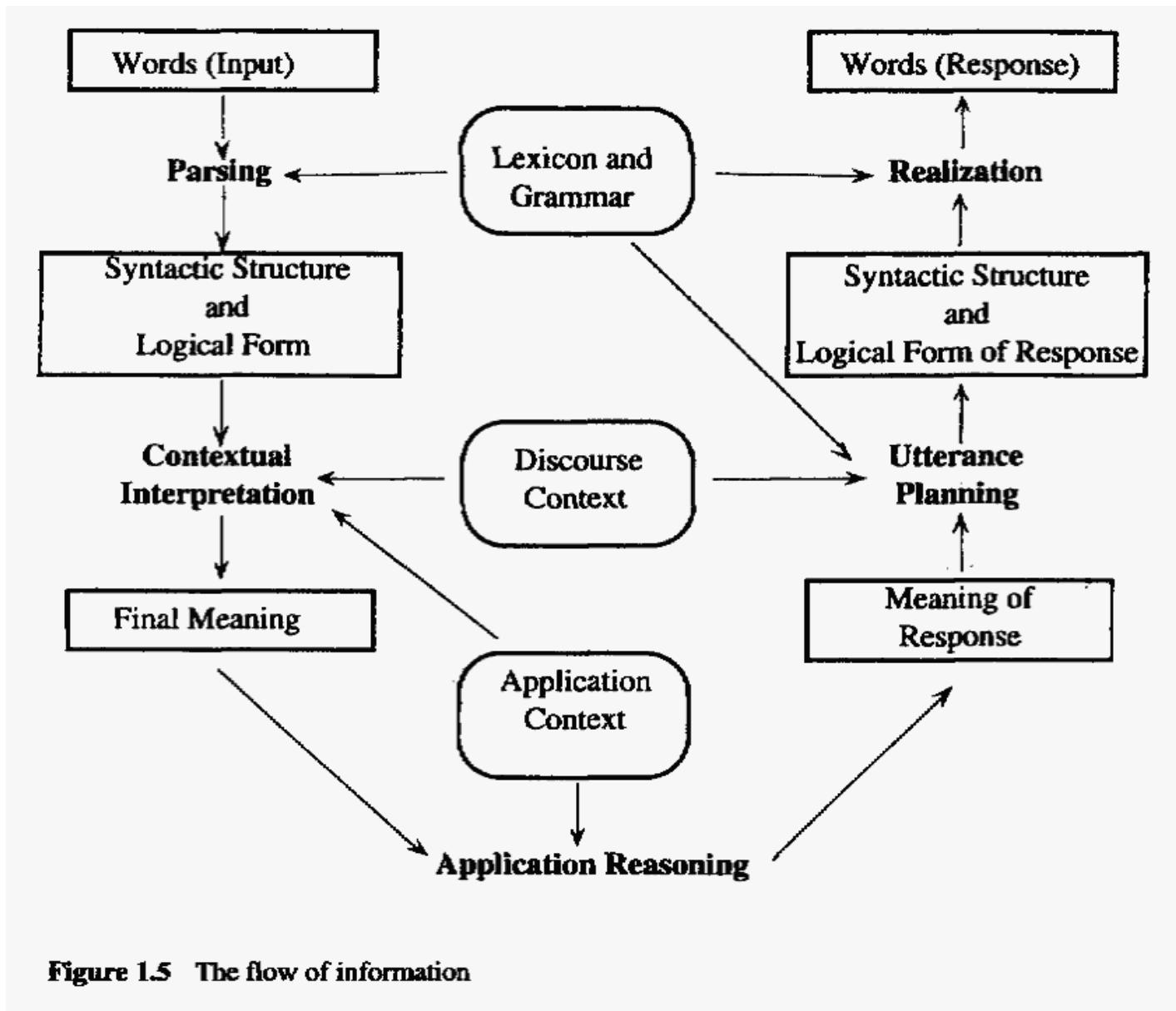


Figure 1.5 The flow of information

Figure 1.5 The flow of information

the event of you visiting relatives. Both of these alternatives are semantically valid, and you would need to determine the appropriate sense by using the contextual mechanism. However, sentence 11 has only one possible semantic interpretation, since museums are not objects that can visit other people; rather they must be visited. In a system with separate syntactic and semantic processing, there would be two syntactic interpretations of sentence 11, one of which the semantic interpreter would eliminate later. If syntactic and semantic processing are combined, however, the system will be able to detect the semantic anomaly as soon as it interprets the phrase "*visiting museums*", and thus will never build the incorrect syntactic structure in the first place. While the savings here seem small, in a realistic application a reasonable sentence may have hundreds of possible syntactic structures, many of which are semantically anomalous.

Continuing through Figure 1.5, the process that transforms the syntactic structure and logical form into a final meaning representation is called contextual processing. This process includes issues such as identifying the objects referred to by noun phrases such as definite descriptions (for example, "*the man*") and pronouns, the analysis of the temporal aspects of the new information conveyed by the sentence, the identification of the speaker's intention (for example, whether "*Can you lift that rock*" is a yes/no question or a request), as well as all the

[Allen 1995 : Chapter 1 - Introduction / 17]

inferential processing required to interpret the sentence appropriately within the application domain. It uses knowledge of the discourse context (determined by the sentences that preceded the current one) and knowledge of the application to produce a final representation.

The system would then perform whatever reasoning tasks are appropriate for the application. When this requires a response to the user, the meaning that must be expressed is passed to the generation component of the system. It uses knowledge of the discourse context, plus information on the grammar and lexicon, to plan the form of an utterance, which then is mapped into words by a realization process. Of course, if this were a spoken language application, the words would not be the final input and output, but rather would be the output of a speech recognizer and the input to a speech synthesizer, as appropriate.

While this text focuses primarily on language understanding, notice that the same levels of knowledge are also used for the generation task as well. For instance, knowledge of syntactic structure is encoded in the grammar. This grammar can be used either to identify the structure of a given sentence or to realize a structure as a sequence of words. A grammar that supports both processes is called a bidirectional grammar. While most researchers agree that bidirectional grammars are the preferred model, in actual practice grammars are often tailored for the understanding task or the generation task. This occurs because different issues are important for each task, and

generally any given researcher focuses just on the problems related to their specific task. But even when the actual grammars differ between understanding and generation, the grammatical formalisms used remain the same.

>> [back](#)

Summary

This book describes computational theories of natural language understanding. The principal characteristic of understanding systems is that they compute representations of the meanings of sentences and use these representations in reasoning tasks. Three principal levels of representation were introduced that correspond to the three main subparts of this book. Syntactic processing is concerned with the structural properties of sentences; semantic processing computes a logical form that represents the context-independent meaning of the sentence; and contextual processing connects language to the application domain.

>> [back](#)

Related Work and Further Readings

A good idea of work in the field can be obtained by reading two articles in Shapiro (1992), under the headings "Computational Linguistics" and "Natural Language Understanding". There are also articles on specialized subareas such as machine translation, natural language interfaces, natural language generation, and so on. Longer surveys on certain areas are also available. Slocum (1985) gives a

[Allen 1995 : Chapter 1 - Introduction / 18]

survey of machine translation, and Perrault and Grosz (1986) give a survey of natural language interfaces.

You can find a description of the ELIZA program that includes the transcript of the dialogue in Figure 1.2 in Weizenbaum (1966). The basic technique of using template matching was developed further in the PARRY system, as described in the paper by Colby in Schank and Colby (1973). That same book also contains descriptions of early natural language systems, including those by Winograd and by Schank. Another important

early system is the LUNAR system, an overview of which can be found in Woods (1977). For another perspective on the AI approach to natural language, refer to the introduction in Winograd (1983).

>> [back](#)

Exercises for Chapter 1

1. (easy) Define a set of data rules for ELIZA that would generate the first seven exchanges in the conversation in Figure 1.2.
2. (easy) Discover all of the possible meanings of the following sentences by giving a paraphrase of each interpretation. For each sentence, identify whether the different meanings arise from structural ambiguity, semantic ambiguity, or pragmatic ambiguity.
 - a. Time flies like an arrow.
 - b. He drew one card.
 - c. Mr. Spook was charged with illegal alien recruitment.
 - d. He crushed the key to my heart.
3. (*easy*) Classify these sentences along each of the following dimensions, given that the person uttering the sentence is responding to a complaint that the car is too cold: (i) syntactically correct or not; (ii) semantically correct or not; (iii) pragmatically correct or not.

- a. The heater are on.
 - b. The tires are brand new.
 - c. Too many windows eat the stew.
4. (*medium*) Implement an ELIZA program that can use the rules that you developed in Exercise 1 and run it for that dialogue. Without adding any more rules, what does your program do on the next few utterances in the conversation in Figure 1.2? How does the program do if you run it in a different context - say, a casual conversation at a bar?

[Allen 1995 : Chapter 1 - Introduction / 19]

Part I: Syntactic Processing

[Allen 1995 : Chapter 2 - An Outline of English Syntax / 20]

As discussed in the introduction, this book divides the task of understanding sentences into three stages. Part I of the book discusses the first stage, syntactic processing. The goal of syntactic processing is to determine the structural components of sentences. It determines, for instance, how a sentence is broken down into phrases, how those phrases are broken down into sub-phrases, and so on, all the way down to the actual structure of the words used. These structural relationships are crucial for determining the meaning of sentences using the techniques described in Parts II and III.

There are two major issues discussed in Part I. The first issue concerns the formalism that is used to specify what sentences are possible in a language. This information is specified by a set of rules called a grammar. We will be concerned both with the general issue of what constitutes good formalisms for writing grammars for natural languages, and with the specific issue of what grammatical rules provide a good account of English syntax. The second issue concerns how to determine the structure of a given sentence once you know the grammar for the language. This process is called parsing. There are many different algorithms for parsing, and this book will consider a sampling of the techniques that are most influential in the field.

Chapter 2 provides a basic background to English syntax for the reader who has not studied linguistics. It introduces the key concepts and distinctions that are common to virtually all syntactic theories. Chapter 3 introduces several formalisms that are in common use for specifying grammars and describes the basic parsing algorithms in detail. Chapter 4 introduces the idea of features, which extend the basic grammatical formalisms and allow many aspects of natural languages to be captured concisely. Chapter 5 then describes some of the more difficult aspects of natural languages, especially the treatment of questions, relative clauses, and other forms of movement phenomena. It shows how the feature systems can be extended so that they can handle these complex sentences. Chapter 6 discusses issues relating to ambiguity resolution. Some techniques are aimed at developing more efficient representations for storing multiple interpretations, while others are aimed at using local information to choose between alternative interpretations while the parsing is in progress. Finally, Chapter 7 discusses a relatively new area of research that uses statistical information derived from analyzing large databases of sentences. This information can be used to identify the most likely classes for ambiguous words and the most likely structural analyses for structurally ambiguous sentences.

>> [back](#)

Allen 1995 : Natural Language Understanding

	<u>Contents</u>	<u>Preface</u>	<u>Introduction</u>	
<u>previous chapter</u>	<u>Part I Syntactic Processing</u>	<u>Part II Semantic Interpretation</u>	<u>Part III - Context / World Knowledge</u>	<u>next chapter</u>
	<u>Appendices</u>	<u>Bibliography</u>		
	<u>Summaries</u>	<u>Further Readings</u>	<u>Exercises</u>	

Chapter 3. Grammars and Parsing

[3.1 Grammars and Sentence Structure](#)

[3.2 What Makes a Good Grammar](#)

[3.3 A Top-Down Parser](#)

[3.4 A Bottom-Up Chart Parser](#)

[3.5 Transition Network Grammars](#)

[o 3.6 Top-Down Chart Parsing](#)

[o 3.7 Finite State Models and Morphological Processing](#)

[o 3.8 Grammars and Logic Programming](#)

[Summary](#)

3.1 Grammars and Sentence Structure

Related Work and Further Readings

Exercises for Chapter 3

[Allen 1995: Chapter 3 - Grammars and Parsing 41]

To examine how the syntactic structure of a sentence can be computed, you must consider two things: the grammar, which is a formal specification of the structures allowable in the language, and the parsing technique, which is the method of analyzing a sentence to determine its structure according to the grammar. This chapter examines different ways to specify simple grammars and considers some fundamental parsing techniques. Chapter 4 then describes the methods for constructing syntactic representations that are useful for later semantic interpretation.

The discussion begins by introducing a notation for describing the structure of natural language and describing some naive parsing techniques for that grammar. The second section describes some characteristics of a good grammar. The third section then considers a simple parsing technique and introduces the idea of parsing as a search process. The fourth section describes a method for building efficient parsers using a structure called a chart. The fifth section then describes an alternative representation of grammars based on transition networks. The remaining sections deal with optional and advanced issues. Section 3.6 describes a top-down chart parser that combines the advantages of top-down and bottom-up approaches. Section 3.7 introduces the notion of finite state transducers and discusses their use in morphological processing. Section 3.8 shows how to encode context-free grammars as assertions in PROLOG, introducing the notion of logic grammars.

>> [back](#)

3.1 Grammars and Sentence Structure

This section considers methods of describing the structure of sentences and explores ways of characterizing all the legal structures in a language. The most common way of representing how a sentence is broken into its major subparts, and how those subparts are broken up in turn, is as a tree. The tree representation for the sentence *John ate the cat* is shown in Figure 3.1. This illustration can be read as follows: The sentence (5) consists of an initial noun phrase (NP) and a verb phrase (VP). The initial noun phrase is made of the simple NAME *John*. The verb phrase is composed of a verb (V) *ate* and an NP, which consists of an article (ART) *the* and a common noun (N) *cat*. In list notation this same structure could be represented as

(S (NP (NAME John))

(VP (V ate)

(NP (ART the)

(N cat))))

Since trees play such an important role throughout this book, some terminology needs to be introduced. Trees are a special form of graph, which are structures consisting of labeled nodes (for example, the nodes are labeled 5, NP, and so on in Figure 3.1) connected by links. They are called trees because they resemble upside-down trees, and much of the terminology is derived from this analogy with actual trees. The node at the top is called the root of the tree, while

[Allen 1995: Chapter 3 - Grammars and Parsing 42]

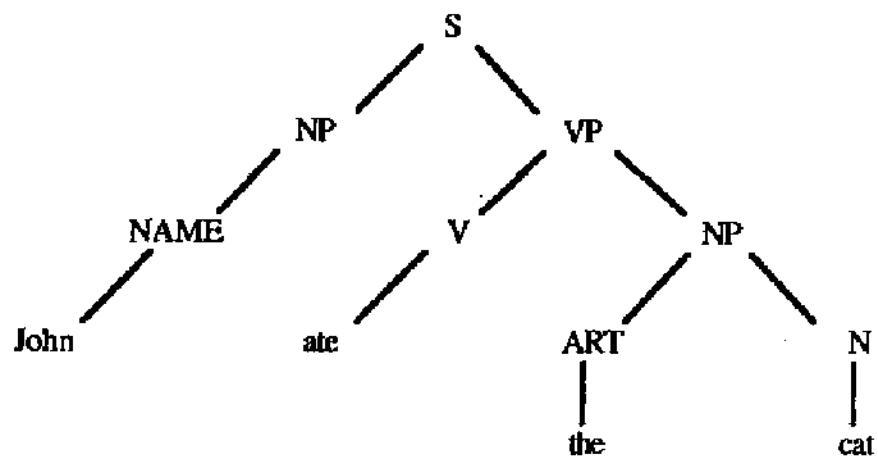


Figure 3.1 A tree representation of *John ate the cat*

Figure 3.1 A tree representation of *John ate the cat*

- | | |
|----------------------------|----------------------------|
| 1. $S \rightarrow NP\ VP$ | 5. $NAME \rightarrow John$ |
| 2. $VP \rightarrow V\ NP$ | 6. $V \rightarrow ate$ |
| 3. $NP \rightarrow NAME$ | 7. $ART \rightarrow the$ |
| 4. $NP \rightarrow ART\ N$ | 8. $N \rightarrow cat$ |

Grammar 3.2 A simple grammar

Grammar 3.2 A simple grammar

the nodes at the bottom are called the leaves. We say a link points from a parent node to a child node. The node labeled S in Figure 3.1 is the parent node of the nodes labeled NP and VP, and the node labeled NP is in turn the parent node of the node labeled NAME. While every child node has a unique parent, a parent may point to many child nodes. An ancestor of a node N is defined as N's parent, or the parent of its parent, and so on. A node is dominated by its ancestor nodes. The root node dominates all other nodes in the tree.

To construct a tree structure for a sentence, you must know what structures are legal for English. A set of rewrite rules describes what tree structures are allowable. These rules say that a certain symbol may be expanded in the

tree by a sequence of other symbols. A set of rules that would allow the tree structure in Figure 3.1 is shown as Grammar 3.2. Rule 1 says that an-S may consist of an NP followed by a VP. Rule 2 says that a VP may consist of a V followed by an NP. Rules 3 and 4 say that an NP may consist of a NAME or may consist of an ART followed by an N. Rules 5 - 8 define possible words for the categories. Grammars consisting entirely of rules with a single symbol on the left-hand side, called the mother, are called context-free grammars (CFGs). CFGs are a very important class of grammars for two reasons: The formalism is powerful enough to describe most of the structure in natural languages, yet it is restricted enough so that efficient parsers can be built to analyze sentences. Symbols that cannot be further decomposed in a grammar, namely the words in the preceding example, are called terminal symbols. The other symbols, such as NP, VP, and S, are called

[Allen 1995: Chapter 3 - Grammars and Parsing 43]

nonterminal symbols. The grammatical symbols such as N and V that describe word categories are called lexical symbols. Of course, many words will be listed under multiple categories. For example, *can* would be listed under V and N.

Grammars have a special symbol called the start symbol. In this book, the start symbol will always be S. A grammar is said to derive a sentence if there is a sequence of rules that allow you to rewrite the start symbol into the sentence. For instance, Grammar 3.2 derives the sentence *John ate the cat*. This can be seen by showing the sequence of rewrites starting from the S symbol, as follows:

S

```
=> NP VP (rewriting S)
=> NAME VP (rewriting NP)
=> John VP (rewriting NAME)
=> John V NP (rewriting VP)
=> John ate NP (rewriting V)
=> John ate ART N (rewriting NP)
=> John ate the N (rewriting ART)
=> John ate the cat (rewriting N)
```

Two important processes are based on derivations. The first is sentence generation, which uses derivations to construct legal sentences. A simple generator could be implemented by randomly choosing rewrite rules, starting from the S symbol, until you have a sequence of words. The preceding example shows that the sentence *John ate the cat* can be generated from the grammar. The second process based on derivations is parsing, which identifies the structure of sentences given a grammar. There are two basic methods of searching. A top-down strategy starts with the S symbol and then searches through different ways to rewrite the symbols until the input sentence is generated, or until all possibilities have been explored. The preceding example demonstrates that *John ate the cat* is a legal sentence by showing the derivation that could be found by this process.

In a bottom-up strategy, you start with the words in the sentence and use the rewrite rules backward to reduce the sequence of symbols until it consists solely of S. The left-hand side of each rule is used to rewrite the symbol on the right-hand side. A possible bottom-up parse of the sentence *John ate the cat* is

```
=> NAME ate the cat (rewriting John)
=> NAME Vthe cat (rewriting ate)
=> NAME V ART cat (rewriting the)
=> NAME V ART N (rewriting cat)
=> NP V ART N (rewriting NAME)
=> NP V NP (rewriting ART N)
=> NP VP (rewriting V NP)
=> S (rewriting NP VP)
```

A tree representation, such as Figure 3.1, can be viewed as a record of the CFG rules that account for the structure of the sentence. In other words, if you

[Allen 1995: Chapter 3 - Grammars and Parsing 44]

kept a record of the parsing process, working either top-down or bottom-up, it would be something similar to the parse tree representation.

>> [back](#)

3.2 What Makes a Good Grammar

In constructing a grammar for a language, you are interested in generality, the range of sentences the grammar analyzes correctly; selectivity, the range of non-sentences it identifies as problematic; and understandability, the simplicity of the grammar itself.

In small grammars, such as those that describe only a few types of sentences, one structural analysis of a sentence may appear as understandable as another, and little can be said as to why one is superior to the other. As you attempt to extend a grammar to cover a wide range of sentences, however, you often find that one analysis is easily extendable while the other requires complex modification. The analysis that retains its simplicity and generality as it is extended is more desirable.

Unfortunately, here you will be working mostly with small grammars and so will have only a few opportunities to evaluate an analysis as it is extended. You can attempt to make your solutions generalizable, however, by keeping in mind certain properties that any solution should have. In particular, pay close attention to the way the sentence is divided into its subparts, called constituents. Besides using your intuition, you can apply a few specific tests, discussed here.

Anytime you decide that a group of words forms a particular constituent, try to construct a new sentence that involves that group of words in a conjunction with another group of words classified as the same type of

constituent. This is a good test because for the most part only constituents of the same type can be conjoined. The sentences in Figure 3.3, for example, are acceptable, but the following sentences are not:

*I ate a hamburger and on the stove.

*I ate a cold hot dog and well burned.

*I ate the hot dog slowly and a hamburger.

To summarize, if the proposed constituent doesn't conjoin in some sentence with a constituent of the same class, it is probably incorrect.

Another test involves inserting the proposed constituent into other sentences that take the same category of constituent. For example, if you say that *John's hitting of Mary* is an NP in *John's hitting of Mary alarmed Sue*, then it should be usable as an NP in other sentences as well. In fact this is true—the NP can be the object of a verb, as in *I cannot explain John's hitting of Mary* as well as in the passive form of the initial sentence *Sue was alarmed by John's hitting of Mary*. Given this evidence, you can conclude that the proposed constituent appears to behave just like other NPs.

[Allen 1995: Chapter 3 - Grammars and Parsing 45]

NP	NP: I ate <i>a hamburger</i> and <i>a hot dog</i> .
VP	VP: I will <i>eat the hamburger</i> and <i>throw away the hot dog</i> .
S	S: <i>I ate a hamburger</i> and <i>John ate a hot dog</i> .
PP	PP: I saw a hot dog <i>in the bag</i> and <i>on the stove</i> .
ADJP	ADJP: I ate a <i>cold</i> and <i>well burned</i> hot dog.
ADVP	ADVP: I ate the hot dog <i>slowly</i> and <i>very carefully</i> .
N	N: I ate a <i>hamburger</i> and <i>hot dog</i> .
V	V: I will <i>cook</i> and <i>burn</i> a hamburger.
AUX	AUX: I <i>can</i> and <i>will</i> eat the hot dog.
ADJ	ADJ: I ate the very <i>cold</i> and <i>burned</i> hot dog (that is, very cold and very burned).

Figure 3.3 Various forms of conjunctions

As another example of applying these principles, consider the two sentences *I looked up John ‘s phone number* and *I looked up John ‘s chimney*. Should these sentences have the identical structure? If so, you would presumably analyze both as subject-verb-complement sentences with the complement in both cases being a PP. That is, *up John ‘s phone number* would be a PP.

When you try the conjunction test, you should become suspicious of this analysis. Conjoining *up John ‘s phone number* with another PP, as in **I looked up John ‘s phone number and in his cupboards*, is certainly bizarre. Note that *I looked up John ‘s chimney and in his cupboards* is perfectly acceptable. Thus apparently the analysis of *up John ‘s phone number* as a PP is incorrect.

Further evidence against the PP analysis is that *up John ‘s phone number* does not seem usable as a PP in any sentences other than ones involving a few verbs such as *look* or *thought*. Even with the verb *look*, an alternative sentence such as **Up John ‘s phone number, I looked* is quite implausible compared to *Up John ‘s chimney, I looked*.

This type of test can be taken further by considering changing the PP in a manner that usually is allowed. In particular, you should be able to replace the NP *John ‘s phone number* by the pronoun *it*. But the resulting sentence, *I looked up it*, could not be used with the same meaning as *I looked up John ‘s phone number*. In fact,

the only way to use a pronoun and retain the original meaning is to use *I looked it up*, corresponding to the form *I looked John's phone number up*.

Thus a different analysis is needed for each of the two sentences. If *up John's phone number* is not a PP, then two remaining analyses may be possible. The VP could be the complex verb *looked up* followed by an NP, or it could consist of three components: the V *looked*, a particle *up*, and an NP. Either of these is a better solution. What types of tests might you do to decide between them?

As you develop a grammar, each constituent is used in more and more different ways. As a result, you have a growing number of tests that can be performed to see if a new analysis is reasonable or not. Sometimes the analysis of a

[Allen 1995: Chapter 3 - Grammars and Parsing 46]

BOX 3.1 Generative Capacity

Grammatical formalisms based on rewrite rules can be compared according to their generative capacity, which is the range of languages that each formalism can describe. This book is concerned with natural languages, but it turns out that no natural language can be characterized precisely enough to define generative capacity. Formal languages, however, allow a precise mathematical characterization.

Consider a formal language consisting of the symbols a , b , c . and d (think of these as words). Then consider a language L_1 that allows any sequence of letters in alphabetical order. For example, abd , ad , bcd , b , and $abcd$ are all legal sentences. To describe this language, we can write a grammar in which the right-hand side of every rule consists of one terminal symbol possibly followed by one nonterminal. Such a grammar is called a regular grammar. For L_1 the grammar would be

$$\begin{array}{llll} S \rightarrow aS_1 & S \rightarrow d & S_1 \rightarrow d & S_3 \rightarrow d \\ S \rightarrow bS_2 & S_1 \rightarrow bS_2 & S_2 \rightarrow cS_3 & \\ S \rightarrow cS_3 & S_1 \rightarrow cS_3 & S_2 \rightarrow d & \end{array}$$

Consider another language, L_2 , that consists only of sentences that have a sequence of a 's followed by an equal number of b 's—that is, ab , $aabb$, $aaabbb$, and so on. You cannot write a regular grammar that can generate L_2 exactly. A context-free grammar to generate L_2 , however, is simple:

$$S \rightarrow a b \qquad S \rightarrow a S b$$

Some languages cannot be generated by a CFG. One example is the language that consists of a sequence of a 's, followed by the same number of b 's, followed by the same number of c 's - that is, abc , $aabbcc$, $aaabbbccc$, and so on. Similarly, no context-free grammar can generate the language that consists of any sequence of letters repeated in the same order twice, such as $ahab$, $abcabc$, $acdabacdacdab$, and so on. There are more general grammatical systems that can generate such sequences, however. One important class is the context-sensitive grammar, which consists of rules of the form

$$\alpha A \beta \rightarrow \alpha \psi \beta$$

where A is a symbol, \tilde{N} and \tilde{O} are (possibly empty) sequences of symbols, and ψ is a nonempty sequence of symbols. Even more general are the type 0 grammars, which allow arbitrary rewrite rules.

Work in formal language theory began with Chomsky (1956). Since the languages generated by regular grammars are a subset of those generated by context-free grammars, which in turn are a subset of those generated by context-sensitive grammars, which in turn are a subset of those generated by type 0 languages, they form a hierarchy of languages (called the Chomsky Hierarchy).

new form might force you to back up and modify the existing grammar. This backward step is unavoidable given the current state of linguistic knowledge. The important point to remember, though, is that when a new rule is proposed for a grammar, you must carefully consider its interaction with existing rules.

>> [back](#)

[Allen 1995: Chapter 3 - Grammars and Parsing 47]

1. S → NP VP
2. NP → ART N
3. NP → ART ADJ N

4. VP → V

5. VP → V NP

Grammar 3.4

3.3 A Top-Down Parser

A parsing algorithm can be described as a procedure that searches through various ways of combining grammatical rules to find a combination that generates a tree that could be the structure of the input sentence. To keep this initial formulation simple, we will not explicitly construct the tree. Rather, the algorithm will simply return a yes or no answer as to whether such a tree could be built. In other words, the algorithm will say whether a certain sentence is accepted by the grammar or not. This section considers a simple top-down parsing method in some detail and then relates this to work in artificial intelligence (AI) on search procedures.

A top-down parser starts with the S symbol and attempts to rewrite it into a sequence of terminal symbols that matches the classes of the words in the input sentence. The state of the parse at any given time can be represented as a list of symbols that are the result of operations applied so far, called the symbol list. For example, the parser starts in the state (S) and after applying the rule S → NP VP the symbol list will be (NP VP). If it then applies the rule NP → ART N, the symbol list will be (ART N VP), and so on.

The parser could continue in this fashion until the state consisted entirely of terminal symbols, and then it could check the input sentence to see if it matched. But this would be quite wasteful, for a mistake made early on (say, in choosing the rule that rewrites S) is not discovered until much later. A better algorithm checks the input as soon as it can. In addition, rather than having a separate rule to indicate the possible syntactic categories for each word, a structure called the lexicon is used to efficiently store the possible categories for each word. For now the lexicon will be very simple. A very small lexicon for use in the examples is

cried: V

dogs: N, V

the: ART

With a lexicon specified, a grammar, such as that shown as Grammar 3.4, need not contain any lexical rules.

Given these changes, a state of the parse is now defined by a pair: a symbol list similar to before and a number indicating the current position in the sentence. Positions fall between the words, with 1 being the position before the first word. For example, here is a sentence with its positions indicated:

[Allen 1995: Chapter 3 - Grammars and Parsing 48]

1 The 2 dogs 3 cried 4

A typical parse state would be

((N VP) 2)

indicating that the parser needs to find an N followed by a VP, starting at position two. New states are generated from old states depending on whether the first symbol is a lexical symbol or not. If it is a lexical symbol, like N in the preceding example, and if the next word can belong to that lexical category, then you can update the state by removing the first symbol and updating the position counter. In this case, since the word *dogs* is listed as an N in the lexicon, the next parser state would be

((VP) 3)

which means it needs to find a VP starting at position 3. If the first symbol is a nonterminal, like VP, then it is rewritten using a rule from the grammar. For example, using rule 4 in Grammar 3.4, the new state would be

((V) 3)

which means it needs to find a V starting at position 3. On the other hand, using rule 5, the new state would be

((V NP) 3)

A parsing algorithm that is guaranteed to find a parse if there is one must systematically explore every possible new state. One simple technique for this is called backtracking. Using this approach, rather than generating a single new state from the state ((VP) 3), you generate all possible new states. One of these is picked to be the next state and the rest are saved as backup states. If you ever reach a situation where the current state cannot lead to a solution, you simply pick a new current state from the list of backup states. Here is the algorithm in a little more detail.

A Simple Top-Down Parsing Algorithm

The algorithm manipulates a list of possible states, called the possibilities list. The first element of this list is the current state, which consists of a symbol list

- and a word position In the sentence, and the remaining elements of the search state are the backup states, each indicating an alternate symbol-list—word-position pair. For example, the possibilities list

((N) 2) ((NAME) 1) ((ADJ N) 1))

indicates that the current state consists of the symbol list (N) at position 2, and that there are two possible backup states: one consisting of the symbol list (NAME) at position 1 and the other consisting of the symbol list (ADJ N) at position 1.

[Allen 1995: Chapter 3 - Grammars and Parsing 49]

Step	Current State	Backup States	Comment
1.	((S) 1)		initial position
2.	((NP VP) 1)		rewriting S by rule I
3.	((ART N VP) 1)		rewriting NP by rules 2 & 3
		((ART ADJN VP) I)	
4.	((N VP) 2)		matching ART with <i>the</i>
		((ART ADJ N VP) 1)	
5.	((VP) 3)		matching N with <i>dogs</i>
		((ART ADJ N VP) 1)	
6.	((V) 3)		rewriting VP by rules 5—8

Step	Current State	Backup States	Comment
		((V NP) 3)	
		((ART ADJ N VP) 1)	
7.			the parse succeeds as V is
			matched to <i>cried</i> , leaving
			an empty grammatical symbol
			list with an empty sentence

Figure 3.5 Top-down depth-first parse of $1 \text{ The } 2 \text{ dogs } 3 \text{ cried } 4$

The algorithm starts with the initial state ((S) 1) and no backup states.

1. Select the current state: Take the first state off the possibilities list and call it C. If the possibilities list is empty, then the algorithm fails (that is, no successful parse is possible).
2. If C consists of an empty symbol list and the word position is at the end of the sentence, then the algorithm succeeds.
3. Otherwise, generate the next possible states.
 - 3.1. If the first symbol on the symbol list of C is a lexical symbol, and the next word in the sentence can be in that class, then create a new state by removing the first symbol from the symbol list and updating the word position, and add it to the possibilities list.
 - 3.2. Otherwise, if the first symbol on the symbol list of C is a non-terminal, generate a new state for each rule in the grammar that can rewrite that nonterminal symbol and add them all to the possibilities list.

Consider an example. Using Grammar 3.4, Figure 3.5 shows a trace of the algorithm on the sentence *The dogs cried*. First, the initial S symbol is rewritten using rule 1 to produce a new current state of ((NP VP) 1) in step 2. The NP is then rewritten in turn, but since there are two possible rules for NP in the grammar, two possible states are generated: The new current state involves (ART N VP) at position 1, whereas the backup state involves (ART ADJ N VP) at position 1. In step 4 a word in category ART is found at position I of the

[Allen 1995: Chapter 3 - Grammars and Parsing 50]

sentence, and the new current state becomes (N VP). The backup state generated in step 3 remains untouched. The parse continues in this fashion to step 5, where two different rules can rewrite VP. The first rule generates the new current state, while the other rule is pushed onto the stack of backup states. The parse completes successfully in step 7, since the current state is empty and all the words in the input sentence have been accounted for.

Consider the same algorithm and grammar operating on the sentence

1 **The** 2 **old** 3 **man** 4 **cried** 5

In this case assume that the word *old* is ambiguous between an ADJ and an N and that the word *man* is ambiguous between an N and a V (as in the sentence *The sailors man the boats*). Specifically, the lexicon is

the: ART

old: ADJ, N

man: N, V

cried: V

The parse proceeds as follows (see Figure 3.6). The initial S symbol is rewritten by rule 1 to produce the new current state of ((NP VP) 1). The NP is rewritten in turn, giving the new state of ((ART N VP) 1) with a backup state of ((ART ADJ N VP) 1). The parse continues, finding *the* as an ART to produce the state ((N VP) 2) and then *old* as an N to obtain the state ((VP) 3). There are now two ways to rewrite the VP, giving us a current state of ((V) 3) and the backup states of ((V NP) 3) and ((ART ADJ N) 1) from before. The word *man* can be parsed as a V, giving the state (04). Unfortunately, while the symbol list is empty, the word position is not at the end of the sentence, so no new state can be generated and a backup state must be used. In the next cycle, step 8, ((V NP) 3) is attempted. Again *man* is taken as a V and the new state ((NP) 4) generated. None of the rewrites of NP yield a successful parse. Finally, in step 12, the last backup state, ((ART ADJ N VP) 1), is tried and leads to a successful parse.

Parsing as a Search Procedure

You can think of parsing as a special case of a search problem as defined in AI. In particular, the top-down parser in this section was described in terms of the following generalized search procedure. The possibilities list is initially set to the start state of the parse. Then you repeat the following steps until you have success or failure:

1. Select the first state from the possibilities list (and remove it from the list).
2. Generate the new states by trying every possible option from the selected state (there may be none if we are on a bad path).

3. Add the states generated in step 2 to the possibilities list.

[Allen 1995: Chapter 3 - Grammars and Parsing 51]

Step	Current State	Backup States	Comment
1.	((S) 1)		
2.	((NP VP) 1)		
3.	((ART N VP) 1)	((ART ADJ N VP) 1)	S rewritten to NP VP NP rewritten producing two new states
4.	((N VP) 2)	((ART ADJ N VP) 1)	
5.	((VP) 3)	((ART ADJ N VP) 1)	the backup state remains
6.	((V) 3)	((V NP) 3) ((ART ADJ N VP) 1)	
7.	(() 4)	((V NP) 3) ((ART ADJ N VP) 1)	
8.	((V NP) 3)	((ART ADJ N VP) 1)	the first backup is chosen
9.	((NP) 4)	((ART ADJ N VP) 1)	
10.	((ART N) 4)	((ART ADJ N) 4) ((ART ADJ N VP) 1)	looking for ART at 4 fails
11.	((ART ADJ N) 4)	((ART ADJ N VP) 1)	fails again
12.	((ART ADJ N VP) 1)		now exploring backup state saved in step 3
13.	((ADJ N VP) 2)		
14.	((N VP) 3)		
15.	((VP) 4)		
16.	((V) 4)	((V NP) 4)	
17.	(() 5)		success!

Figure 3.6 A top-down parse of 1 The 2 old 3 man 4 cried 5

Figure 3.6 A top-down parse of $_1 \text{The}_2 \text{old man}_4 \text{cried}_5$

For a depth-first strategy, the possibilities list is a stack. In other words, step 1 always takes the first element off the list, and step 3 always puts the new states on the front of the list, yielding a last-in first-out (LIFO) strategy.

In contrast, in a breadth-first strategy the possibilities list is manipulated as a queue. Step 3 adds the new positions onto the end of the list, rather than the beginning, yielding a first-in first-out (FIFO) strategy.

[Allen 1995: Chapter 3 - Grammars and Parsing 52]

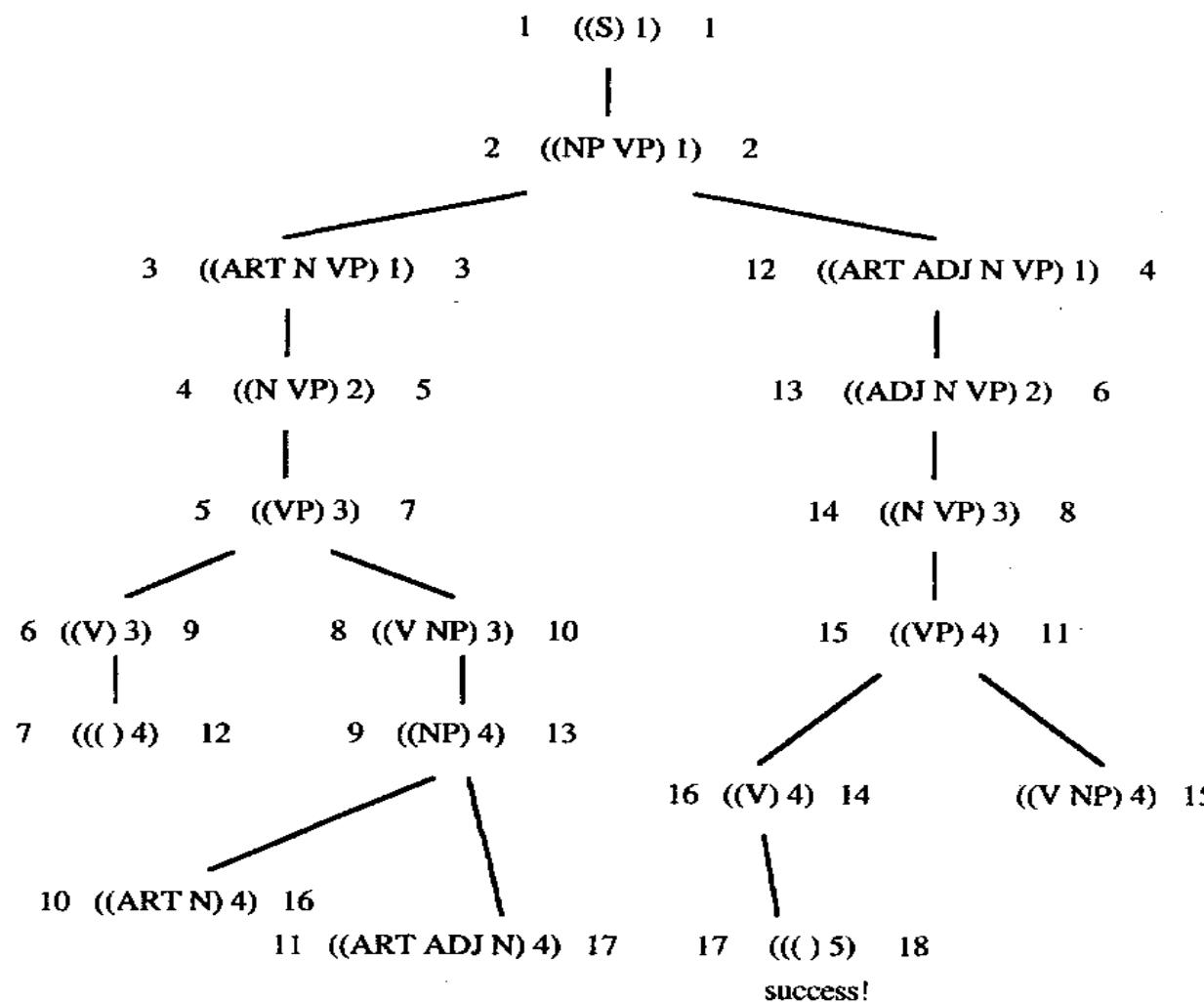


Figure 3.7 Search tree for two parse strategies (depth-first strategy on left; breadth-first on right)

Figure 3.7 Search tree for two parse strategies (depth-first strategy on left; breadth-first on right)

We can compare these search strategies using a tree format, as in Figure 3.7, which shows the entire space of parser states for the last example. Each node in the tree represents a parser state, and the sons of a node are the possible moves from that state. The number beside each node records when the node was selected to be processed by the algorithm. On the left side is the order produced by the depth-first strategy, and on the right side is the order produced by the breadth-first strategy. Remember, the sentence being parsed is

1 **The** 2 **old** 3 **man** 4 **cried** 5

The main difference between depth-first and breadth-first searches in this simple example is the order in which the two possible interpretations of the first NP are examined. With the depth-first strategy, one interpretation is considered and expanded until it fails; only then is the second one considered. With the breadth-first strategy, both interpretations are considered alternately, each being

[Allen 1995: Chapter 3 - Grammars and Parsing 53]

expanded one step at a time. In this example, both depth-first and breadth-first searches found the solution but searched the space in a different order. A depth-first search often moves quickly to a solution but in other cases may spend considerable time pursuing futile paths. The breadth-first strategy explores each possible solution to a certain depth before moving on. In this particular example

-the depth-first strategy found the solution in one less step than the breadth-first. (The state in the bottom right-hand side of Figure 3.7 was not explored by the depth-first parse.)

In certain cases it is possible to put these simple search strategies into an infinite loop. For example, consider a left-recursive rule that could be a first account of the possessive in English (as in the NP *the man's coat*):

NP → NP 's N

With a naive depth-first strategy, a state starting with the nonterminal NP would be rewritten to a new state beginning with NP s N. But this state also begins with an NP that could be rewritten in the same way. Unless an explicit check were incorporated into the parser, it would rewrite NPs forever! The breadth-first strategy does better with left-recursive rules, as it tries all other ways to rewrite the original NP before coming to the newly generated state with the new NP. But with an ungrammatical sentence it would not terminate because it would rewrite the NP forever while searching for a solution. For this reason, many systems prohibit left-recursive rules from the grammar.

Many parsers built today use the depth-first strategy because it tends to minimize the number of backup states needed and thus uses less memory and requires less bookkeeping.

>> [back](#)

3.4 A Bottom-Up Chart Parser

The main difference between top-down and bottom-up parsers is the way the grammar rules are used. For example, consider the rule

NP → ART ADJ N

In a top-down system you use the rule to find an NP by looking for the sequence ART ADJ N. In a bottom-up parser you use the rule to take a sequence ART ADJ N that you have found and identify it as an NP. The basic operation in bottom-up parsing then is to take a sequence of symbols and match it to the right-hand side of the rules. You could build a bottom-up parser simply by formulating this matching process as a search process. The state would simply consist of a symbol list, starting with the words in the sentence. Successor states could be generated by exploring all possible ways to

- rewrite a word by its possible lexical categories
- replace a sequence of symbols that matches the right-hand side of a grammar rule by its left-hand side symbol

[Allen 1995: Chapter 3 - Grammars and Parsing 54]

1. **S** → **N PVP**
2. **NP** → **ART ADJ N**
3. **NP** → **ART N**
4. **NP** → **ADJ N**
5. **VP** → **AUX VP**
6. **VP** → **V NP**

Grammar 3.8 A simple context-free grammar

Unfortunately, such a simple implementation would be prohibitively expensive, as the parser would tend to try the same matches again and again, thus duplicating much of its work unnecessarily. To avoid this problem, a data

structure called a chart is introduced that allows the parser to store the partial results of the matching it has done so far so that the work need not be reduplicated.

Matches are always considered from the point of view of one constituent, called the key. To find rules that match a string involving the key, look for rules that start with the key, or for rules that have already been started by earlier keys and require the present key either to complete the rule or to extend the rule. For instance, consider Grammar 3.8.

Assume you are parsing a sentence that starts with an ART. With this ART as the key, rules 2 and 3 are matched because they start with ART. To record this for analyzing the next key, you need to record that rules 2 and 3 could be continued at the point after the ART. You denote this fact by writing the rule with a dot (o), indicating what has been seen so far. Thus you record

2' . NP -> ART o ADJ N

3' . NP -> ART o N

If the next input key is an ADJ, then rule 4 may be started, and the modified rule 2 may be extended to give

2'' . NP -> ART ADJ o N

The chart maintains the record of all the constituents derived from the sentence so far in the parse. It also maintains the record of rules that have matched partially but are not complete. These are called the active arcs. For example, after seeing an initial ART followed by an ADS in the preceding example, you would have the chart shown in Figure 3.9. You should interpret this figure as follows. There are two completed constituents on the chart: ART1 from position 1 to 2 and ADJ1 from position 2 to 3. There are four active arcs indicating possible

constituents. These are indicated by the arrows and are interpreted as follows (from top to bottom). There is a potential NP starting at position 1, which needs an ADJ starting at position 2. There is another potential NP starting at position 1, which needs an N starting at position 2. There is a potential NP

[Allen 1995: Chapter 3 - Grammars and Parsing 55]

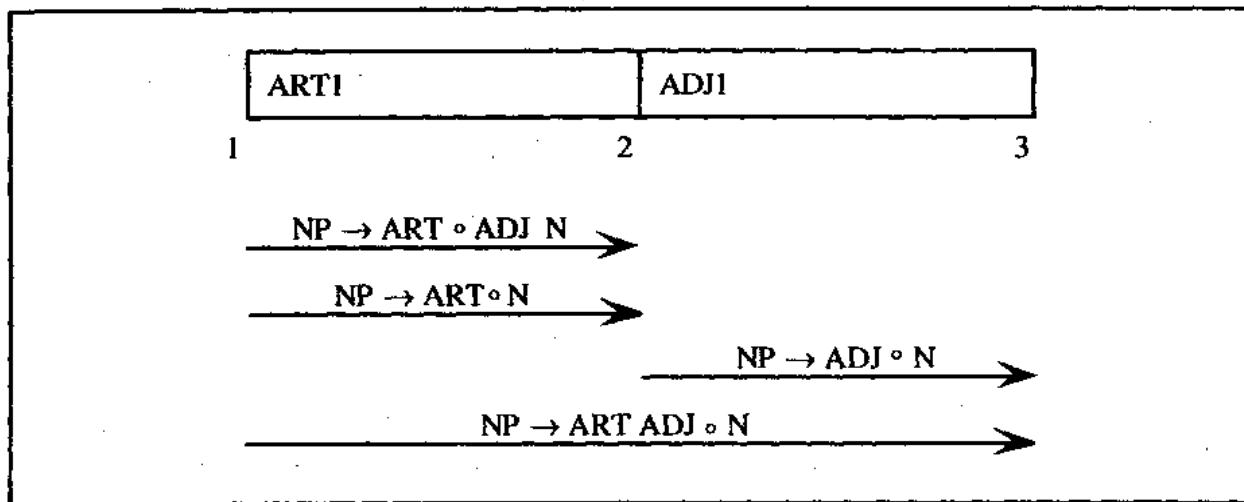


Figure 3.9 The chart after seeing an ADJ in position 2

Figure 3.9 The chart after seeing an ADJ in position 2

To add a constituent C from position p_1 to p_2 :

1. Insert C into the chart from position p_1 to p_2 .
2. For any active arc of the form $X \rightarrow X_1 \dots \circ C \dots X_n$ from position p_0 to p_1 ,
add a new active arc $X \rightarrow X_1 \dots C \circ \dots X_n$ from position p_0 to p_2 .
3. For any active arc of the form $X \rightarrow X_1 \dots X_n \circ C$ from position p_0 to p_1 ,
then add a new constituent of type X from p_0 to p_2 to the agenda.

Figure 3.10 The arc extension algorithm

Figure 3.10 The arc extension algorithm

starting at position 2 with an ADS, which needs an N starting at position 3. Finally, there is a potential NP starting at position 1 with an ART and then an ADJ, which needs an N starting at position 3.

The basic operation of a chart-based parser involves combining an active arc with a completed constituent. The result is either a new completed constituent or a new active arc that is an extension of the original active arc. New completed constituents are maintained on a list called the agenda until they themselves are added to the chart. This process is defined more precisely by the arc extension algorithm shown in Figure 3.10. Given this algorithm, the bottom-up chart parsing algorithm is specified in Figure 3.11.

As with the top-down parsers, you may use a depth-first or breadth-first search strategy, depending on whether the agenda is implemented as a stack or a queue. Also, for a full breadth-first strategy, you would need to read in the entire input and add the interpretations of the words onto the agenda before starting the algorithm. Let us assume a depth-first search strategy for the following example.

Consider using the algorithm on the sentence *The large can can hold the water* using Grammar 3.8 with the following lexicon:

[Allen 1995: Chapter 3 - Grammars and Parsing 56]

Do until there is no input left:

1. If the agenda is empty, look up the interpretations for the next word in the input and add them to the agenda.
2. Select a constituent from the agenda (let's call it constituent C from position p1 to p2).
3. For each rule in the grammar of form $\mathbf{x} \rightarrow \mathbf{c} \ \mathbf{x}_1 \dots \mathbf{x}_n$, add an active arc of form $\mathbf{x} \rightarrow \mathbf{c} \circ \mathbf{c} \circ \mathbf{x}_1 \dots \mathbf{x}_n$ from position p1 to p2.
4. Add C to the chart using the arc extension algorithm above.

Figure 3.11 A bottom-up chart parsing algorithm

the: ART
 large: ADS
 can: N,AUX,V
 hold: N,V
 water: N, V

To best understand the example, draw the chart as it is extended at each step of the algorithm. The agenda is initially empty, so the word *the* is read and a constituent ARTL placed on the agenda.

Entering ART1: (the from 1 to 2)

Adds active arc NP -> ART o ADJ N from 1 to 2

Adds active arc NP -> ART o N from 1 to 2

Both these active arcs were added by step 3 of the parsing algorithm and were derived from rules 2 and 3 in the grammar, respectively. Next the word *large* is read and a constituent ADJ1 is created.

Entering ADJ1: (large from 2 to 3)

Adds arc NP -> ADS o N from 2 to 3

Adds arc NP -> ART ADJ o N from 1 to 3

The first arc was added in step 3 of the algorithm. The second arc added here is an extension of the first active arc that was added when ART1 was added to the chart using the arc extension algorithm (step 4).

The chart at this point has already been shown in Figure 3.9. Notice that active arcs are never removed from the chart. For example, when the arc NP -4 ART o ADS N from 1 to 2 was extended, producing the arc from 1 to 3,

both arcs remained on the chart. This is necessary because the arcs could be used again in a different way by another interpretation.

For the next word, *can*, three constituents, Ni, AUX1, and V1 are created for its three interpretations.

[Allen 1995: Chapter 3 - Grammars and Parsing 57]

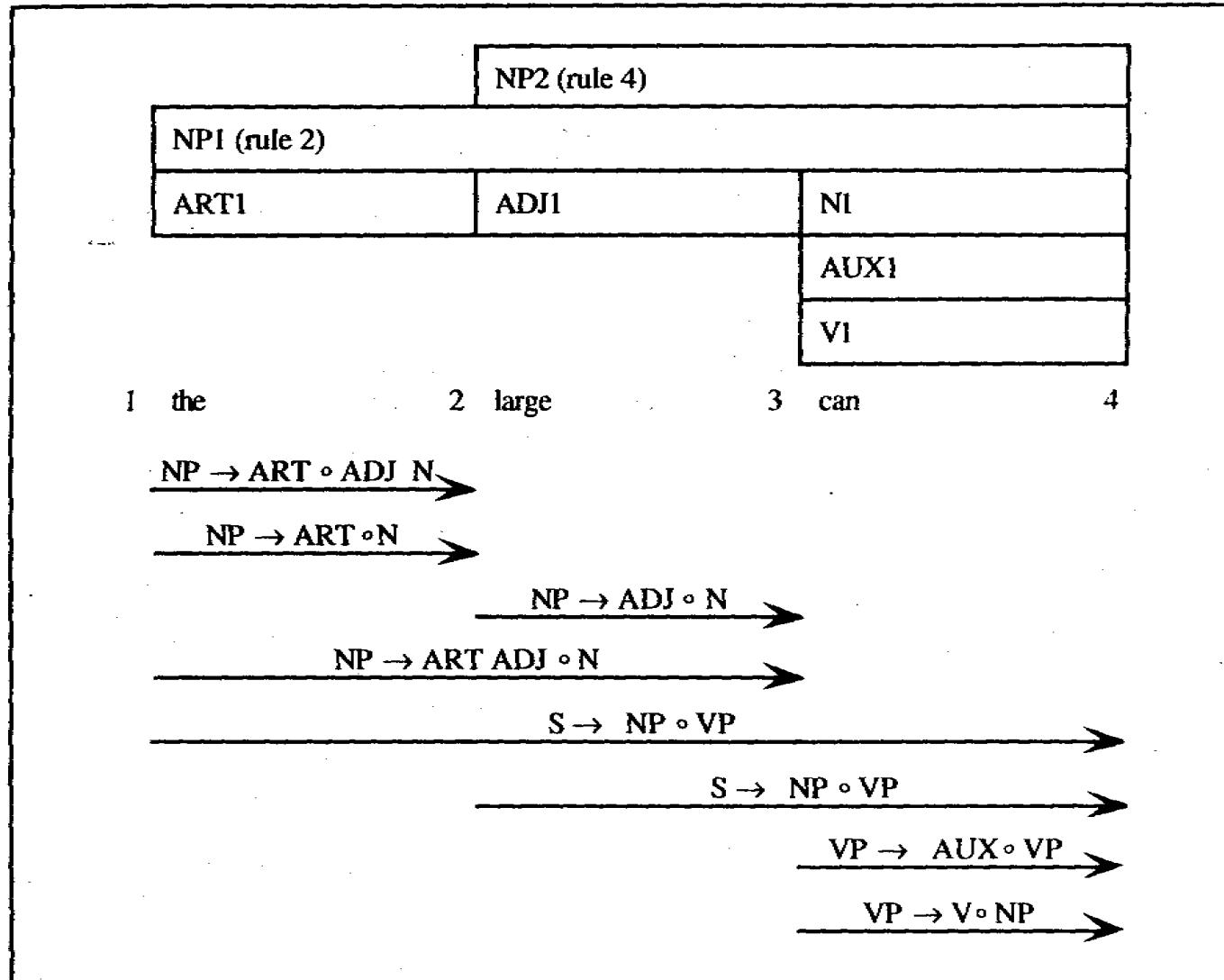


Figure 3.12 After parsing *the large can*

Figure 3.12 After parsing *the large can*

Entering N1 (*can* from 3 to 4)

No active arcs are added in step 2, but two are completed in step 4 by the arc extension algorithm, producing two NPs that are added to the agenda: The first, an NP from 1 to 4, is constructed from rule 2, while the second, an NP from 2 to 4, is constructed from rule 4. These NPs are now at the top of the agenda.

Entering NP1 :an NP (*the large can* from 1 to 4)

Adding active arc S -> NP o VP from 1 to 4

Entering NP2: an NP (*large can* from 2 to 4)

Adding arc S -> NP o VP from 2 to 4

Entering AUX1: (*can* from 3 to 4)

Adding arc VP -> AUX o VP from 3 to 4

Entering V1: (*can* from 3 to 4)

Adding arc VP -> V o NP from 3 to 4

The chart is shown in Figure 3.12, which illustrates all the completed constituents (NP2, NP1, ART1, ADJ1, N1, AUX1, V1) and all the uncompleted active arcs entered so far. The next word is *can* again. and N2, AUX, and V2 are created.

[Allen 1995: Chapter 3 - Grammars and Parsing 58]

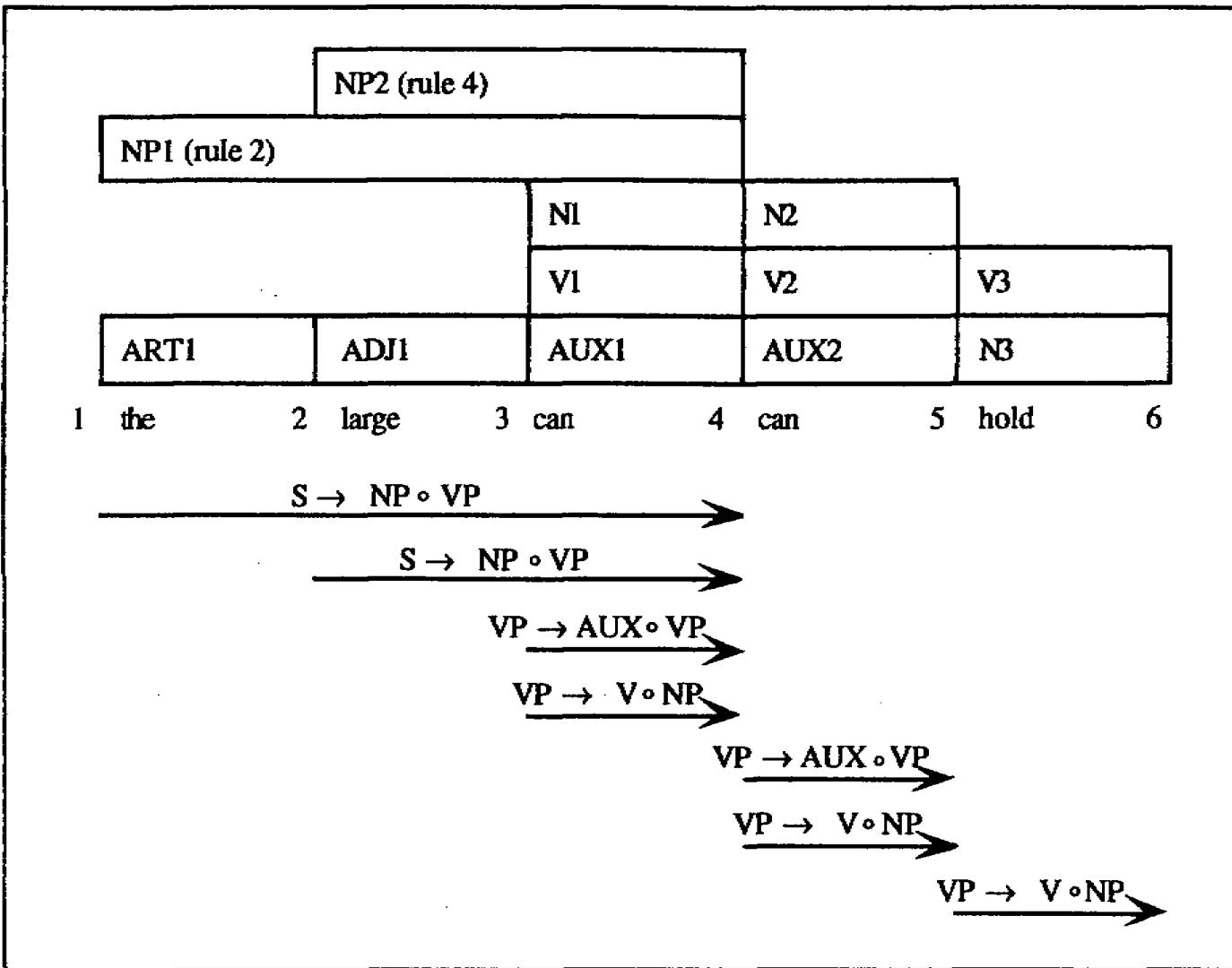


Figure 3.13 The chart after adding *hold*, omitting arcs generated for the first NP

Figure 3.13 The chart after adding *hold*, omitting arcs generated for the first NP

Entering N2: (can from 4 to 5, the second can)

Adds no active arcs

Entering AUX2: (can from 4 to 5)

Adds arc VP -> AUX o VP from 4 to 5

Entering V2: (can from 4 to 5)

Adds arc VP -> V o NP from 4 to 5

The next word is *hold*, and N3 and V3 are created.

Entering N3: (hold from 5 to 6)

Adds no active arcs

Entering V3: (hold from 5 to 6)

Adds arc VP -> V o NP from 5 to 6

The chart in Figure 3.13 shows all the completed constituents built so far, together with all the active arcs, except for those used in the first NP.

Entering ART2: (*the* from 6 to 7)

Adding arc NP -> ART o ADJ N from 6 to 7

Adding arc NP -> ART o N from 6 to 7

[Allen 1995: Chapter 3 - Grammars and Parsing 59]

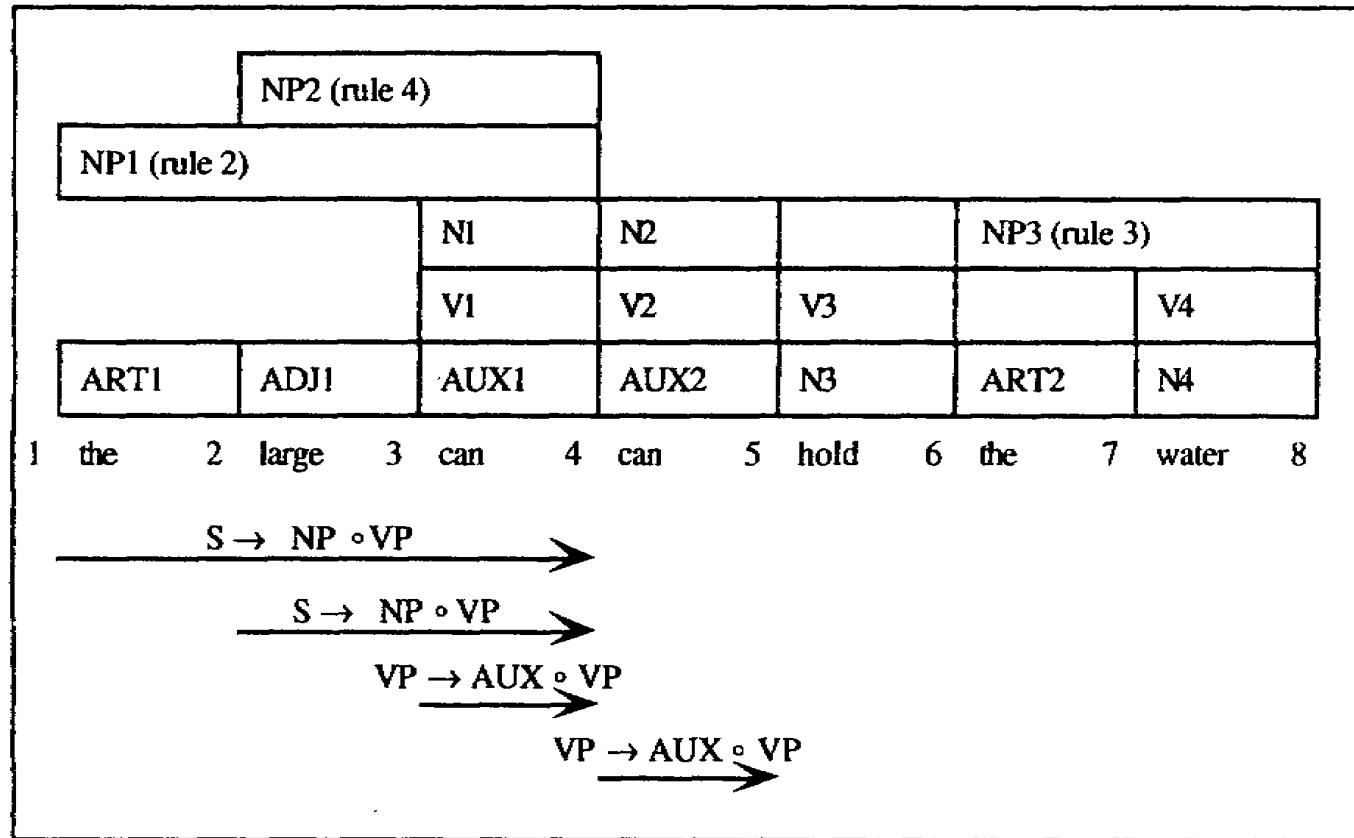


Figure 3.14 The chart after all the NPs are found, omitting all but the crucial active arcs

Figure 3.14 The chart after all the NPs are found, omitting all but the crucial active arcs

Entering N4: (*water* from 7 to 8)

No active arcs added in step 3

An NP, NP3, from 6 to 8 is pushed onto the agenda, by completing arc
NP → ART o N from 6 to 7

Entering NP3: (*the water* from 6 to 8)

A VP, VP1, from 5 to 8 is pushed onto the agenda, by completing VP →
V o NP from 5 to 6

Adds arc S → NP o VP from 6 to 8

The chart at this stage is shown in Figure 3.14, but only the active arcs to be used in the remainder of the parse are shown.

Entering VP1: (*hold the water* from 5 to 8)

A VP, VP2, from 4 to 8 is pushed onto the agenda, by completing

VP -> AUX o VP from 4 to S

Entering VP2: (*can hold the water* from 4 to 8)

An S, S1, is added from 1 to 8, by completing

arcS -> NP o VP from 1 to 4

A VP, VP3, is added from 3 to 8, by completing

arc VP -> AUX o VP from 3 to 4

An S, S2, is added from 2 to 8, by completing

arc S -> NP o VP from 2 to 4

Since you have derived an S covering the entire sentence, you can stop successfully. If you wanted to find all possible interpretations for the sentence,

[Allen 1995: Chapter 3 - Grammars and Parsing 60]

S1 (rule 1 with NP1 and VP2)						
S2 (rule 1 with NP2 and VP2)						
VP3 (rule 5 with AUX1 and VP2)						
NP2 (rule 4)		VP2 (rule 5)				
NP1 (rule 2)			VP1 (rule 6)			
		N1	N2		NP3 (rule 3)	
		V1	V2	V3		V4
ART1	ADJ1	AUX1	AUX2	N3	ART2	N4
1 the	2 large	3 can	4 can	5 hold	6 the	7 water

Figure 3.15 The final chart

Figure 3.15 The final chart

you would continue parsing until the agenda became empty. The chart would then contain as many S structures covering the entire set of positions as there were different structural interpretations. In addition, this representation of the entire set of structures would be more efficient than a list of interpretations, because the different S structures might share common subparts represented in the chart only once. Figure 3.15 shows the final chart.

Efficiency Considerations

Chart-based parsers can be considerably more efficient than parsers that rely only on a search because the same constituent is never constructed more than once. For instance, a pure top-down or bottom-up search strategy could require up to C^n operations to parse a sentence of length n, where C is a constant that depends on the specific algorithm you use. Even if C is very small, this exponential complexity rapidly makes the algorithm unusable. A chart-based parser, on the other hand, in the worst case would build every possible constituent between every possible pair of positions. This allows us to show that it has a worst-case complexity of K^*n^3 , where n is the length of the sentence and K is a constant depending on the algorithm. Of course, a chart parser involves more work in each step, so K will be larger than C. To contrast the two approaches, assume that C is 10 and that K is a hundred times worse, 1000. Given a sentence of 12 words, the brute force search might take 10^{12} operations (that is, 1,000,000,000,000), whereas the chart parser would take $1000 * 12^3$ (that is, 1,728,000). Under these assumptions, the chart parser would be up to 500,000 times faster than the brute force search on some examples!

>> [back](#)

[Allen 1995: Chapter 3 - Grammars and Parsing 61]

3.5 Transition Network Grammars

So far we have examined only one formalism for representing grammars, namely context-free rewrite rules. Here we consider another formalism that is useful in a wide range of applications. It is based on the notion of a transition network consisting of nodes and labeled arcs. One of the nodes is specified as the initial state, or start state. Consider the network named NP in Grammar 3.16, with the initial state labeled NP and each arc labeled with a word category. Starting at the initial state, you can traverse an arc if the current word in the sentence is in the category on the arc. If the arc is followed, the current word is updated to the next word. A phrase is a legal NP if there is a path from the node NP to a pop arc (an arc labeled pop) that accounts for every word in the phrase. This network recognizes the same set of sentences as the following context-free grammar:

NP → ART NP1

NP1 → ADJ NP1

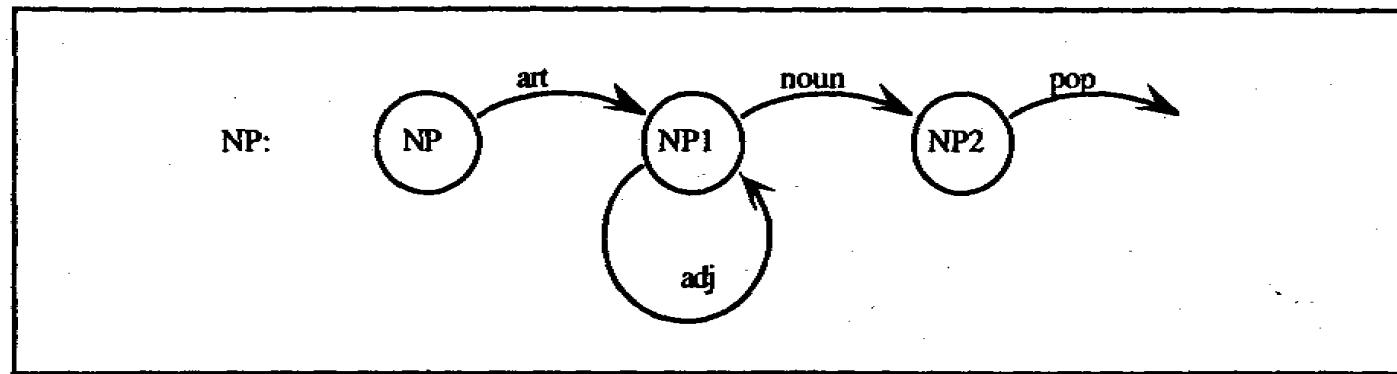
NP1 → N

Consider parsing the NP *a purple cow* with this network. Starting at the node NP, you can follow the arc labeled art, since the current word is an article—namely, *a*. From node NP1 you can follow the arc labeled adj using the adjective *purple*, and finally, again from NP1, you can follow the arc labeled noun using the noun *cow*. Since you have reached a pop arc, *a purple cow* is a legal NP.

Simple transition networks are often called finite state machines (FSMs). Finite state machines are equivalent in expressive power to regular grammars (see Box 3.2), and thus are not powerful enough to describe all languages that can be described by a CFG. To get the descriptive power of CFGs, you need a notion of recursion in the network grammar. A recursive transition network (RTN) is like a simple transition network, except that it allows arc labels to refer to other networks as well as word categories. Thus, given the NP network in Grammar 3.16, a network for simple English sentences can be expressed as shown in Grammar 3.17. Uppercase labels refer to networks. The arc from S to Si can be followed only if the NP network can be successfully traversed to a pop arc. Although not shown in this example, RTNs allow true recursion—that is, a network might have an arc labeled with its own name.

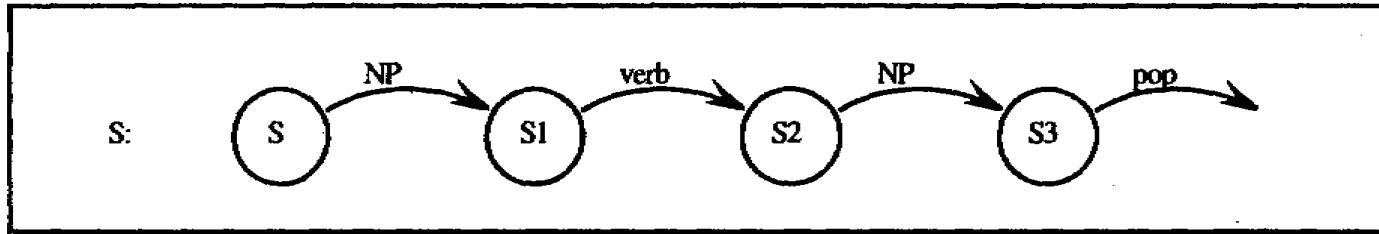
Consider finding a path through the S network for the sentence *The purple cow ate the grass*. Starting at node 5, to follow the arc labeled NP, you need to traverse the NP network. Starting at node NP, traverse the network as before for the input *the purple cow*. Following the pop arc in the NP network, return to the S network and traverse the arc to node S 1. From node S 1 you follow the arc labeled verb using the word *ate*. Finally, the arc labeled NP can be followed if you can traverse the NP network again. This time the remaining input consists of the words *the grass*. You follow the arc labeled art and then the arc labeled noun in the NP network; then take the pop arc from node NP2 and then another pop from node S3. Since you have traversed the network and used all the words in the sentence, *The purple cow ate the grass* is accepted as a legal sentence.

[Allen 1995: Chapter 3 - Grammars and Parsing 62]



Grammar 3.16

Grammar 3.16



Grammar 3.17

Grammar 3.17

Arc Type	Example	Arc Type Example How Used
CAT	noun	succeeds only if current word is of the named category
WRD	of	succeeds only if current word is identical to the label
PUSH	NP	succeeds only if named network can be successfully traversed

Arc Type	Example	Arc Type Example How Used
JUMP	jump	always succeeds
POP	pop	succeeds and signals the successful end of the network

Figure 3.18 The arc labels for RTNs

In practice, RTN systems incorporate some additional arc types that are useful but not formally necessary; Figure 3.18 summarizes the arc types, together with the notation that will be used in this book to indicate these arc types. According to this terminology, arcs that are labeled with networks are called push arcs, and arcs labeled with word categories are called cat arcs. In addition, an arc that can always be followed is called a jump arc.

Top-Down Parsing with Recursive Transition Networks

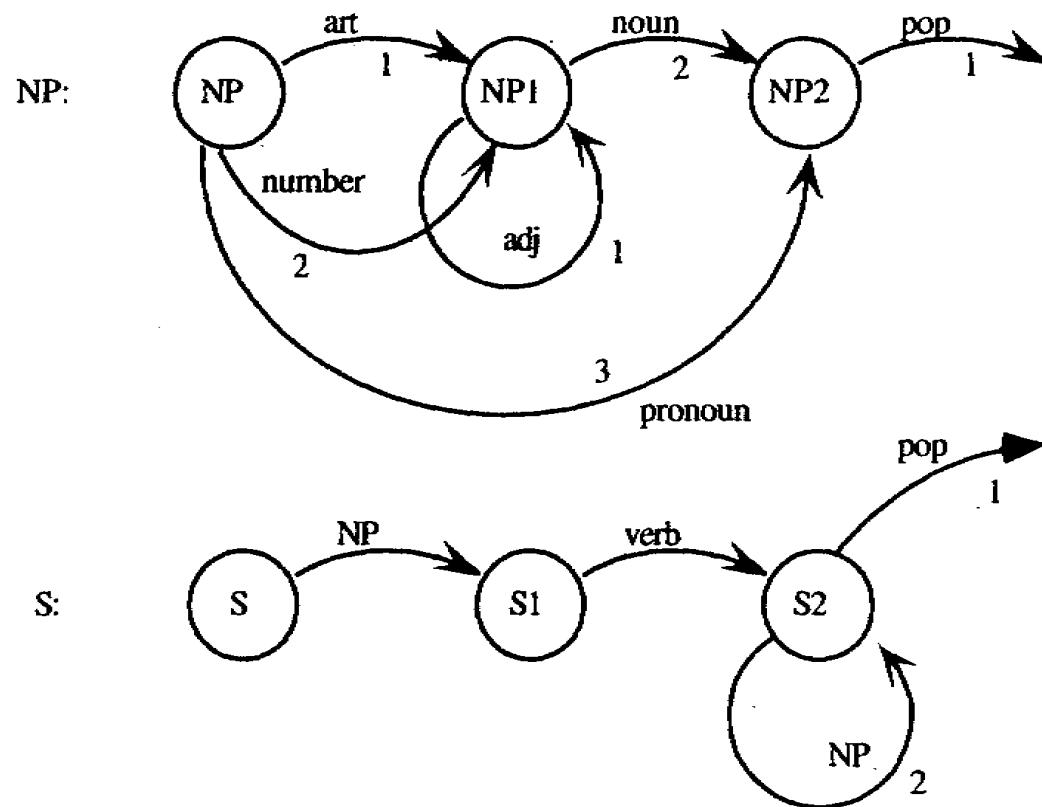
An algorithm for parsing with RTNs can be developed along the same lines as the algorithms for parsing CFGs. The state of the parse at any moment can be represented by the following:

current position - a pointer to the next word to be parsed.

current node - the node at which you are located in the network.

return points - a stack of nodes in other networks where you will continue if you pop from the current network.

[Allen 1995: Chapter 3 - Grammars and Parsing 63]



Grammar 3.19

Grammar 3.19

First, consider an algorithm for searching an RTN that assumes that if you can follow an arc, it will be the correct one in the final parse. Say you are in the middle of a parse and know the three pieces of information just cited. You can leave the current node and traverse an arc in the following cases:

Case 1:

If arc names word category and next word in sentence is in that category,

Then (1) update current position to start at the next word; (2) update current node to the destination of the arc.

Case 2:

If arc is a push arc to a network N,

Then

- (1) add the destination of the arc onto return points;
- (2) update current node to the starting node in network N.

Case 3:

If arc is a pop arc and return points list is not empty,

Then (1) remove first return point and make it current node.

Case 4:

If arc is a pop arc, return points list is empty and there are no words left,

Then (1) parse completes successfully.

Grammar 3.19 shows a network grammar. The numbers on the arcs simply indicate the order in which arcs will be tried when more than one arc leaves a node.

[Allen 1995: Chapter 3 - Grammars and Parsing 64]

Step	Current Node	Current Position	Return Points	Arc to be Followed	Comments
1.	(S,	1,	NIL)	S/1	initial position
2.	(NP,	1,	(S1))	NP/1	followed push arc to NP network, to return ultimately to S1
3.	(NP1,	2,	(S1))	NP1/1	followed arc NP1/1 (<i>the</i>)
4.	(NP1,	3,	(S1))	NP1/2	followed arc NP1/1 (<i>old</i>)
5.	(NP2,	4,	(S1))	NP2/2	followed arc NP1/2 (<i>man</i>) since NP1/1 is not applicable
6.	(S1,	4,	NIL)	S1/1	the pop arc gets us back to S1
7.	(S2,	5,	NIL)	S2/1	followed arc S2/1 (<i>cried</i>)
8.					parse succeeds on pop arc from S2

Figure 3.20 A trace of a top-down parse

Figure 3.20 A trace of a top-down parse

Figure 3.20 demonstrates that the grammar accepts the sentence

1 The 2 old 3 man 4 cried 5

by showing the sequence of parse states that can be generated by the algorithm. In the trace, each arc is identified by the name of the node that it leaves plus the number identifier. Thus arc S/1 is the arc labeled 1 leaving the S node. If you start at node 5, the only possible arc to follow is the push arc NP. As specified in case 2 of the algorithm, the new parse state is computed by setting the current node to NP and putting node S1 on the return points list. From node NP, arc NP/1 is followed and, as specified in case 1 of the algorithm, the input is checked for a word in category art. Since this check succeeds, the arc is followed and the current position is updated (step 3). The parse continues in this manner to step 5, when a pop arc is followed, causing the current node to be reset to S1 (that is, the NP arc succeeded). The parse succeeds after finding a verb in step 6 and following the pop arc from the S network in step 7.

In this example the parse succeeded because the first arc that succeeded was ultimately the correct one in every case. However, with a sentence like *The green faded*, where *green* can be an adjective or a noun, this algorithm would fail because it would initially classify *green* as an adjective and then not find a noun following. To be able to recover from such failures, we save all possible backup states as we go along, just as we did with the CFG top-down parsing algorithm.

Consider this technique in operation on the following sentence:

1 One 2 saw 3 the 4 man 5

The parser initially attempts to parse the sentence as beginning with the NP *one saw*, but after failing to find a verb, it backtracks and finds a successful parse starting with the NP *one*. The trace of the parse is shown in Figure 3.21, where at

[Allen 1995: Chapter 3 - Grammars and Parsing 65]

	Step	Current State	Arc to be Followed	Backup States
	1.	(S, 1, NIL)	S/1	NIL
	2.	(NP, 1, (S1))	NP/2 (& NP/3 for backup)	NIL

	Step	Current State	Arc to be Followed	Backup States
	3.	(NP1, 2, (S1))	NP1/2	(NP2, 2, (S1))
	4.	(NP2, 3, (S1))	NP2/1	(NP2, 2, (S1))
	5.	(S1, 3, NIL)	no arc can be followed	(NP2, 2, (S1))
	6.	(NP2, 2, (S1))	NP2/1	NIL
	7.	(S1, 2, NIL)	S1/1	NIL
	8.	(S2, 3, NIL)	S2/2	NIL
	9.	(NP, 3, (S2))	NP/1	NIL
	10.	(N7PI, 4, (S2))	NP1/2	NIL

	Step	Current State	Arc to be Followed	Backup States
	11.	(NP2, 5, (S2))	NP2/1	NIL
	12.	(S2, 5, NIL)	S2/1	NIL
	13.	parse succeeds		NIL

Figure 3.21 A top-down RTN parse with backtracking

each stage the current parse state is shown in the form of a triple (current node, current position, return points), together with possible states for backtracking. The figure also shows the arcs used to generate the new state and backup states.

This trace behaves identically to the previous example except in two places. In step 2, two arcs leaving node NP could accept the word *one*. Arc NP/2 classifies *one* as a number and produces the next current state. Arc NP/3

classifies it as a pronoun and produces a backup state. This backup state is actually used later in step 6 when it is found that none of the arcs leaving node S 1 can accept the input word *the*.

Of course, in general, many more backup states are generated than in this simple example. In these cases there will be a list of possible backup states. Depending on how this list is organized, you can produce different orderings on when the states are examined.

An RTN parser can be constructed to use a chart-like structure to gain the advantages of chart parsing. In RTN systems, the chart is often called the well-formed substring table (WFST). Each time a pop is followed, the constituent is placed on the WFST, and every time a push is found, the WFST is checked before the subnetwork is invoked. If the chart contains constituent(s) of the type being pushed for, these are used and the subnetwork is not reinvoked. An RTN using a WFST has the same complexity as the chart parser described in the last section: K^*n^3 , where n is the length of the sentence.

>> [back](#)

o 3.6 Top-Down Chart Parsing

So far, you have seen a simple top-down method and a bottom-up chart-based method for parsing context-free grammars. Each of the approaches has its advantages and disadvantages. In this section a new parsing method is presented that

[Allen 1995: Chapter 3 - Grammars and Parsing 66]

actually captures the advantages of both. But first, consider the pluses and minuses of the approaches.

Top-down methods have the advantage of being highly predictive. A word might be ambiguous in isolation, but if some of those possible categories cannot be used in a legal sentence, then these categories may never even be considered. For example, consider Grammar 3.8 in a top-down parse of the sentence *The can holds the water*, where *can* may be an AUX, V. or N, as before.

The top-down parser would rewrite (5) to (NP VP) and then rewrite the NP to produce three possibilities, (ART ADJ N VP), (ART N VP), and (ADJ N VP). Taking the first, the parser checks if the first word, *the*, can be an ART, and then if the next word, *can*, can be an ADJ, which fails. Trying the next possibility, the parser checks *the* again, and then checks if *can* can be an N, which succeeds. The interpretations of *can* as an auxiliary and a main verb are never considered because no syntactic tree generated by the grammar would ever predict an AUX or V in this position. In contrast, the bottom-up parser would have considered all three interpretations of *can* from the start—that is, all three would be added to the chart and would combine with active arcs. Given this argument, the top-down approach seems more efficient.

On the other hand, consider the top-down parser in the example above needed to check that the word *the* was an ART twice, once for each rule. This reduplication of effort is very common in pure top-down approaches and becomes a serious problem, and large constituents may be rebuilt again and again as they are used in different rules. In contrast, the bottom-up parser only checks the input once, and only builds each constituent exactly once. So by this argument, the bottom-up approach appears more efficient.

You can gain the advantages of both by combining the methods. A small variation in the bottom-up chart algorithm yields a technique that is predictive like the top-down approaches yet avoids any reduplication of work as in the bottom-up approaches.

As before, the algorithm is driven by an agenda of completed constituents and the arc extension algorithm, which combines active arcs with constituents when they are added to the chart. While both use the technique of extending arcs with constituents, the difference is in how new arcs are generated from the grammar. In the bottom-up approach, new active arcs are generated whenever a completed constituent is added that could be the first constituent of the right-hand side of a rule. With the top-down approach, new active arcs are generated

whenever a new active arc is added to the chart, as described in the top-down arc introduction algorithm shown in Figure 3.22. The parsing algorithm is then easily stated, as is also shown in Figure 3.22.

Consider this new algorithm operating with the same grammar on the same sentence as in Section 3.4, namely *The large can can hold the water*. In the initialization stage, an arc labeled $S \rightarrow o \text{ NP VP}$ is added. Then, active arcs for each rule that can derive an NP are added: $\text{NP} \rightarrow o \text{ ART ADJ N}$, $\text{NP} \rightarrow o \text{ ART N}$,

[Allen 1995: Chapter 3 - Grammars and Parsing 67]

Top-Down Arc Introduction Algorithm

To add an arc $S \rightarrow C_1 \dots o C_1 \dots C_n$ ending at position j, do the following:

For each rule in the grammar of form $C_i \rightarrow X_1 \dots X_k$, recursively add the new arc $C_i \rightarrow o X_1 \dots X_k$ from position j to j.

Top-Down Chart Parsing Algorithm

Initialization: For every rule in the grammar of form $S \rightarrow X_1 \dots X_k$, add an arc labeled $S \rightarrow o X_1 \dots X_k$ using the arc introduction algorithm.

Parsing: Do until there is no input left:

1. If the agenda is empty, look up the interpretations of the next word and add them to the agenda.
2. Select a constituent from the agenda (call it constituent C).
3. Using the arc extension algorithm, combine C with every active arc on the chart. Any new constituents are added to the agenda.
4. For any active arcs created in step 3, add them to the chart using the top-down arc introduction algorithm.

Figure 3.22 The top-down arc introduction and chart parsing algorithms

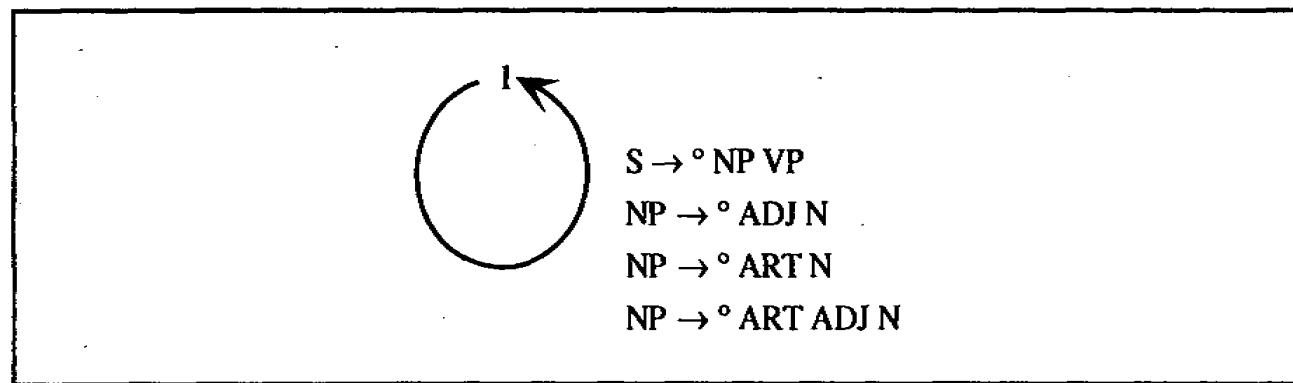


Figure 3.23 The initial chart

Figure 3.23 The initial chart

and $NP \rightarrow {}^o ADJ N$ are all added from position 1 to 1. Thus the initialized chart is as shown in Figure 3.23. The trace of the parse is as follows:

Entering ART1 (*the*) from 1 to 2

Two arcs can be extended by the arc extension algorithm

NP → ART o N from 1 to 2

NP → ART o ADJ N from 1 to 2

Entering ADJ1 (*large*) from 2 to 3

One arc can be extended

NP → ART ADJ o N from 1 to 3

Entering AUX1 (*can*) from 3 to 4

No activity, constituent is ignored

Entering V1 (*can*) from 3 to 4

No activity, constituent is ignored

[Allen 1995: Chapter 3 - Grammars and Parsing 68]

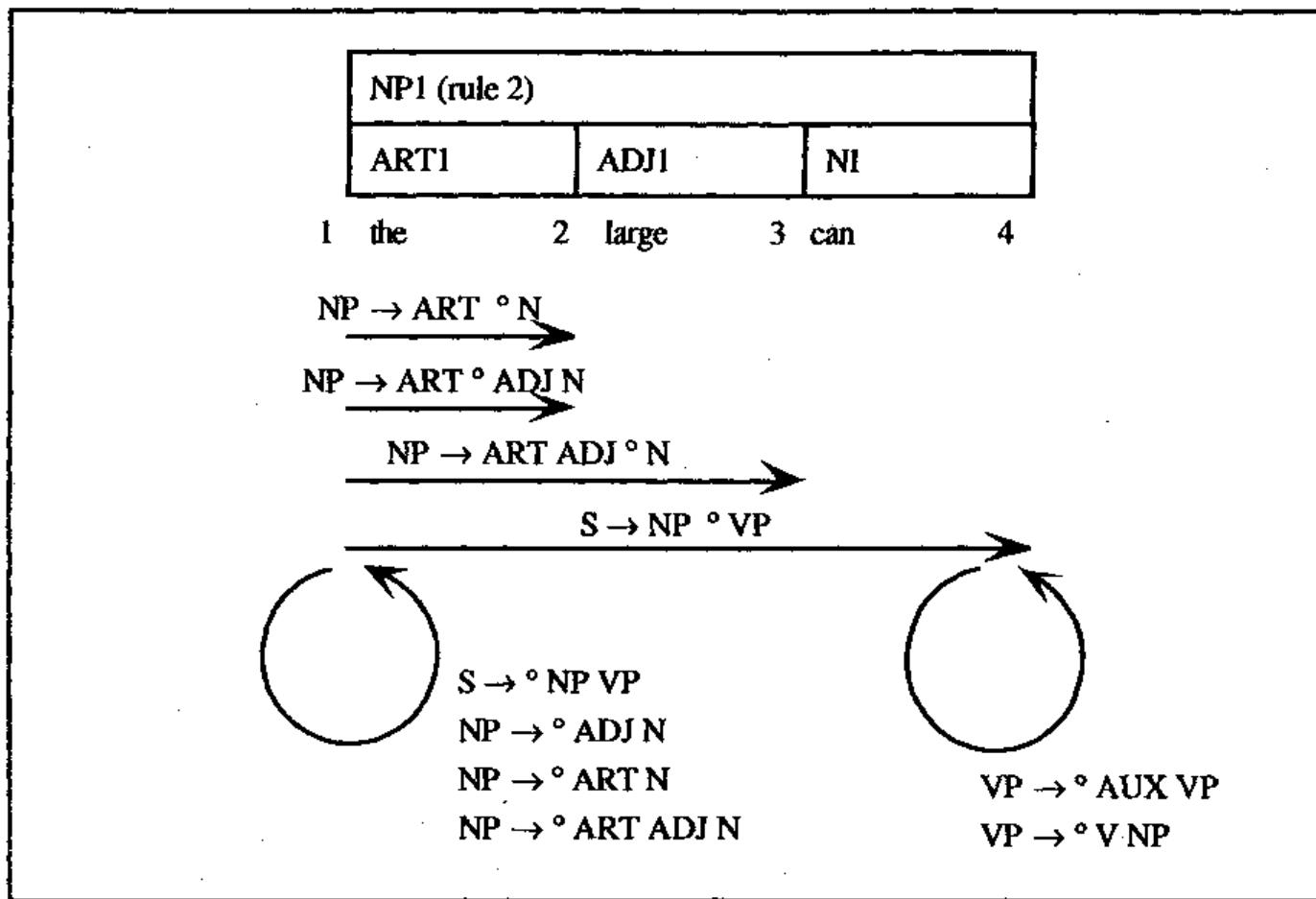


Figure 3.24 The chart after building the first NP

Figure 3.24 The chart after building the first NP

Entering N1 (*can*) from 3 to 4

One arc extended and completed yielding

NP1 from 1 to 4 (*the large can*)

Entering NP1 from 1 to 4

One arc can be extended

S → NP o VP from 1 to 4

Using the top-down rule (step 4), new active arcs are added for VP

VP → o AUX VP from 4 to 4

VP → o V NP from 4 to 4

At this stage, the chart is as shown in Figure 3.24. Compare this with Figure 3.10. It contains fewer completed constituents since only those that are allowed by the top-down filtering have been constructed.

The algorithm continues, adding the three interpretations of *can* as an AUX, V, and N. The AUX reading extends the VP \rightarrow C AUX VP arc at position 4 and adds active arcs for a new VP starting at position 5. The V reading extends the VP 0 V NP arc and adds active arcs for an NP starting at position 5. The N reading does not extend any arc and so is ignored. After the two readings of *hold* (as an N and V) are added, the chart is as shown in Figure 3.25. Again, compare with the corresponding chart for the bottom-up parser in Figure 3.13. The rest of the sentence is parsed similarly, and the final chart is shown in Figure 3.26. In comparing this to the final chart produced by the bottom-up parser (Figure 3.15), you see that the number of constituents generated has dropped from 21 to 13.

[Allen 1995: Chapter 3 - Grammars and Parsing 69]

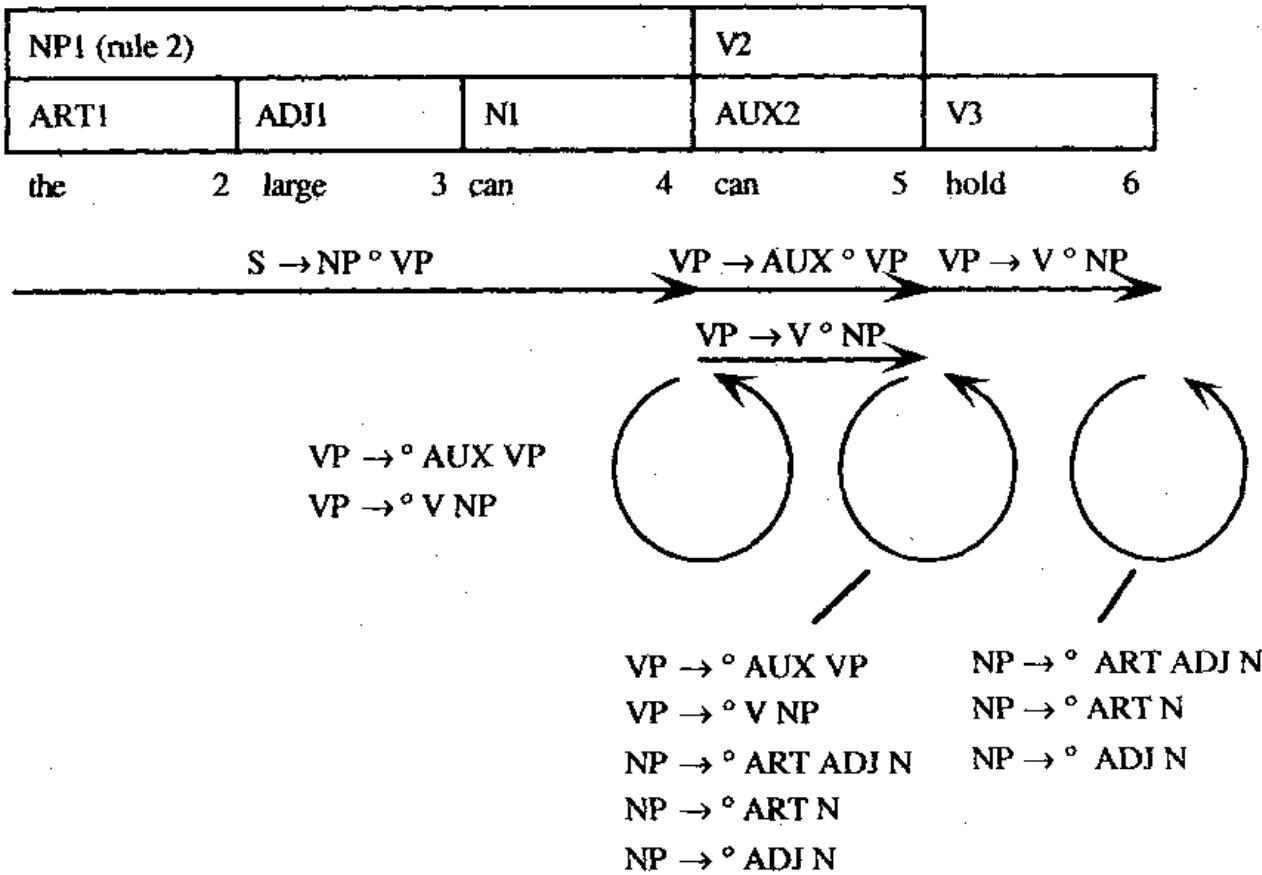


Figure 3.25 The chart after adding *hold*, omitting arcs generated for the first NP

Figure 3.25 The chart after adding *hold*, omitting arcs generated for the first NP

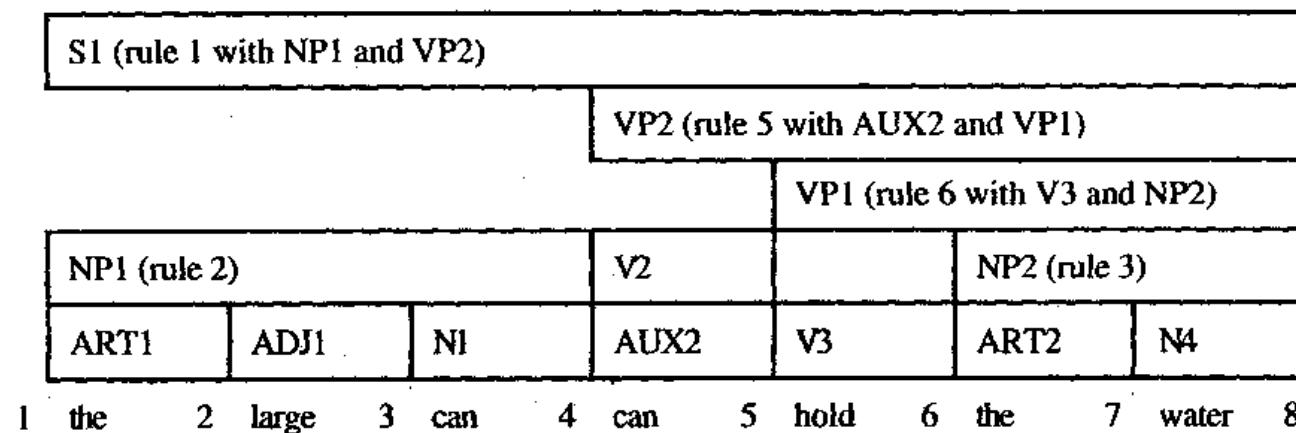


Figure 3.26 The final chart for the top-down filtering algorithm

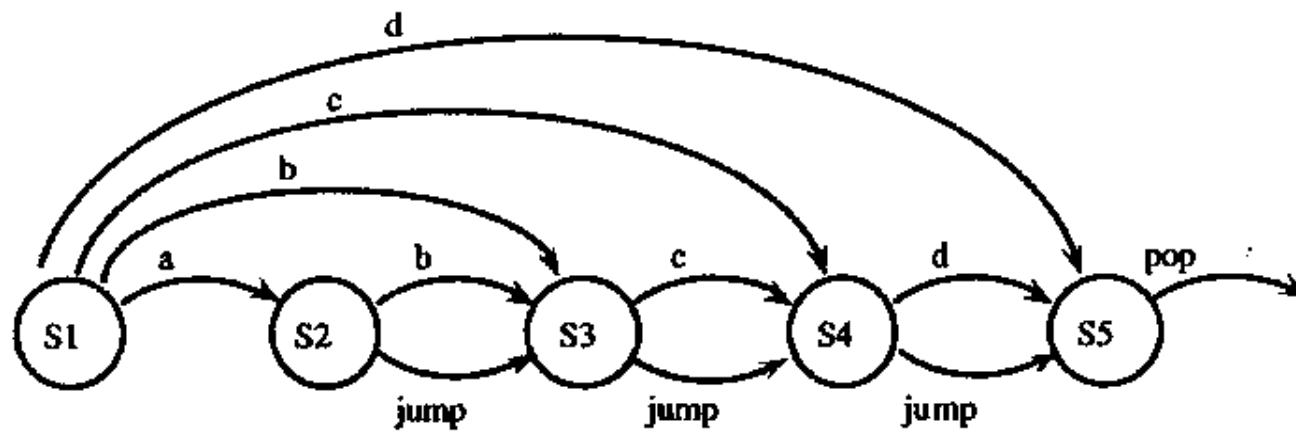
Figure 3.26 The final chart for the top-down filtering algorithm

While it is not a big difference here with such a simple grammar, the difference can be dramatic with a sizable grammar.

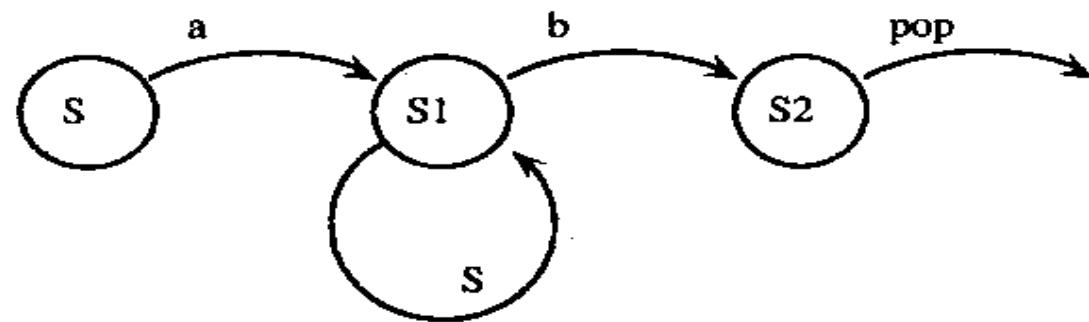
It turns out in the worst-case analysis that the top-down chart parser is not more efficient than the pure bottom-up chart parser. Both have a worst-case complexity of K^*n^3 for a sentence of length n . In practice, however, the top-down method is considerably more efficient for any reasonable grammar.

BOX 3.2 Generative Capacity of Transition Networks

Transition network systems can be classified by the types of languages they can describe. In fact, you can draw correspondences between various network systems and rewrite-rule systems. For instance, simple transition networks (that is, finite state machines) with no push arcs are expressively equivalent to regular grammars—that is, every language that can be described by a simple transition network can be described by a regular grammar, and vice versa. An FSM for the first language described in Box 3.1 is



Recursive transition networks, on the other hand, are expressively equivalent to context-free grammars. Thus an RTN can be converted into a CFG and vice versa. A recursive transition network for the language consisting of a number of a's followed by an equal number of b's is



>> [back](#)

o 3.7 Finite State Models and Morphological Processing

Although in simple examples and small systems you can list all the words allowed by the system, large vocabulary systems face a serious problem in representing the lexicon. Not only are there a large number of words, but each word may combine with affixes to produce additional related words. One way to address this problem is to preprocess the input sentence into a sequence of morphemes. A word may consist of single morpheme, but often a word consists of a root form plus an affix. For instance, the word *eaten* consists of the root form *eat* and the suffix *-en*, which indicates the past participle form. Without any preprocessing, a lexicon would have to list all the forms of *eat*, including *eats*, *eating*, *ate*, and *eaten*. With preprocessing, there would be one morpheme *eat* that may combine with suffixes such as *-ing*, *-s*, and *-en*, and one entry for the irregular form *ate*. Thus the lexicon would only need to store two entries (*eat* and

[Allen 1995: Chapter 3 - Grammars and Parsing 71]

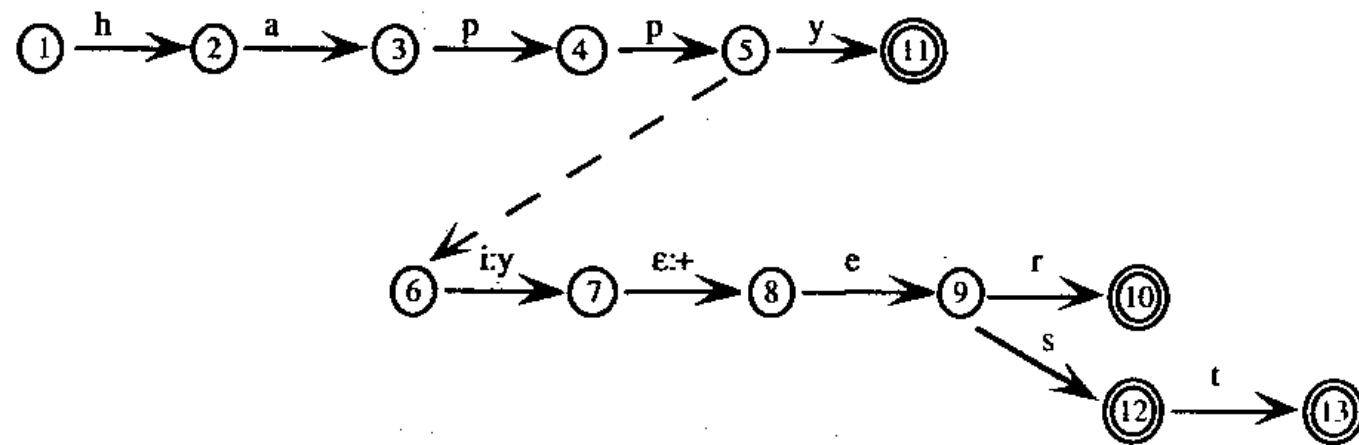


Figure 3.27 A simple FST for the forms of *happy*

Figure 3.27 A simple FST for the forms of *happy*

ate) rather than four. Likewise the word *happiest* breaks down into the root form *happy* and the suffix *-est*, and thus does not need a separate entry in the lexicon. Of course, not all forms are allowed; for example, the word *seed* cannot be decomposed into a root form *se* (or *see*) and a suffix *-ed*. The lexicon would have to encode what forms are allowed with each root.

One of the most popular models is based on finite state transducers (FSTs), which are like finite state machines except that they produce an output given an input. An arc in an FST is labeled with a pair of symbols. For example, an arc labeled $i:y$ could only be followed if the current input is the letter I and the output is the letter y . FSTs can be used to concisely represent the lexicon and to transform the surface form of words into a sequence of morphemes. Figure 3.27 shows a simple FST that defines the forms of the word *happy* and its derived forms. It transforms the word *happier* into the sequence *happy +er* and *happiest* into the sequence *happy +est*.

Arcs labeled by a single letter have that letter as both the input and the output. Nodes that are double circles indicate success states, that is, acceptable words. Consider processing the input word *happier* starting from state 1. The upper network accepts the first four letters, *happ*, and copies them to the output. From state 5 you could accept a *y* and have a complete word, or you could jump to state 6 to consider affixes. (The dashed link, indicating a jump, is not formally necessary but is useful for showing the break between the processing of the root form and the processing of the suffix.) For the word *happier*, you must jump to state 6. The next letter must be an *i*, which is transformed into a *y*. This is followed by a transition that uses no input (the empty symbol r) and outputs a plus sign. From state 8, the input must be an *e*, and the output is also *e*. This must be followed by an *r* to get to state 10, which is encoded as a double circle indicating a possible end of word (that is, a success state for the FST). Thus this FST accepts the appropriate forms and outputs the desired sequence of morphemes.

The entire lexicon can be encoded as an FST that encodes all the legal input words and transforms them into morphemic sequences. The FSTs for the

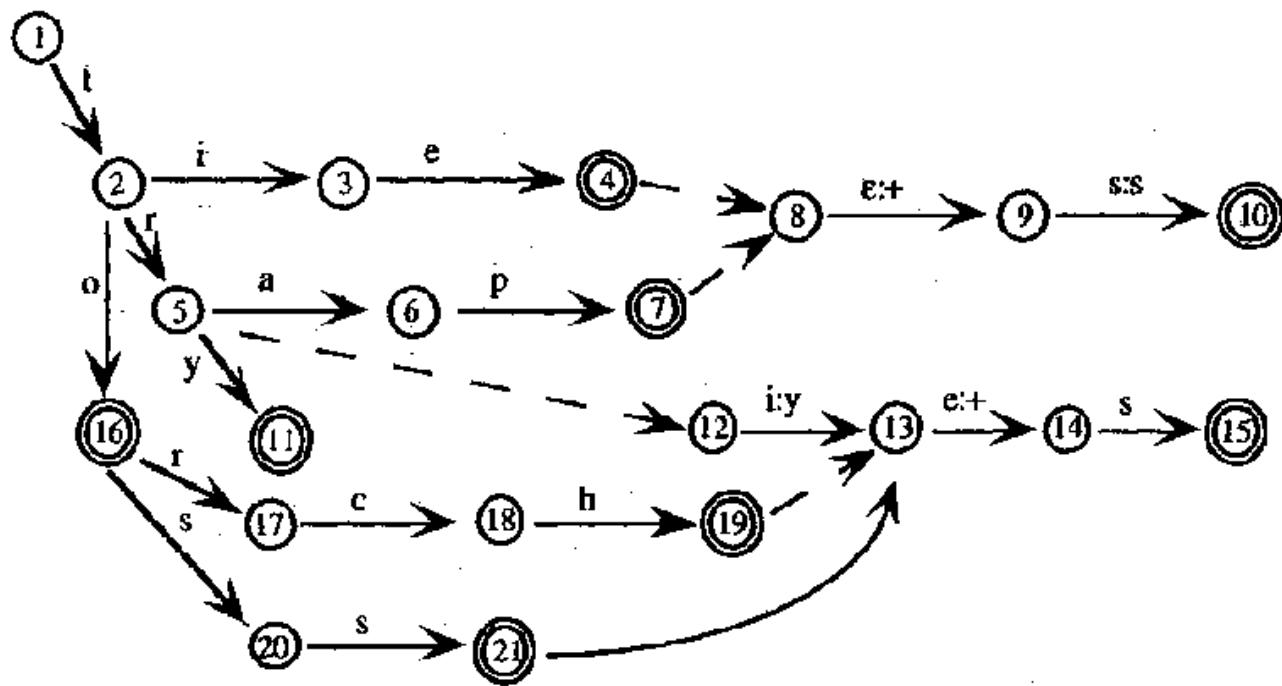


Figure 3.28 A fragment of an FST defining some nouns (singular and plural)

Figure 3.28 A fragment of an FST defining some nouns (singular and plural)

different suffixes need only be defined once, and all root forms that allow that suffix can point to the same node. Words that share a common prefix (such as *torch*, *toss*, and *a*,) also can share the same nodes, greatly reducing the size of the network. The FST in Figure 3.28 accepts the following words, which all start with *t*: *tie* (state 4), *ties* (10), *trap* (7), *traps* (10), *try* (11), *tries* (15), *to* (16), *torch* (19), *torches* (15), *toss* (21), and *tosses* (15). In addition, it outputs the appropriate sequence of morphemes.

Note that you may pass through acceptable states along the way when processing a word. For instance, with the input *toss* you would pass through state 15, indicating that *to* is a word. This analysis is not useful, however, because if *to* was accepted then the letters sr would not be accounted for.

Using such an FST, an input sentence can be processed into a sequence of morphemes. Occasionally, a word will be ambiguous and have multiple different decompositions into morphemes. This is rare enough, however, that we will ignore this minor complication throughout the book.

>> [back](#)

o 3.8 Grammars and Logic Programming

Another popular method of building a parser for CFGs is to encode the rules of the grammar directly in a logic programming language such as PROLOG. It turns out that the standard PROLOG interpretation algorithm uses exactly the same search strategy as the depth-first top-down parsing algorithm, so all that is needed is a way to reformulate context-free grammar rules as clauses in PROLOG. Consider the following CFG rule:

S → NP VP

[Allen 1995: Chapter 3 - Grammars and Parsing 73]

This rule can be reformulated as an axiom that says, "A sequence of words is a legal S if it begins with a legal NP that is followed by a legal VP." If you number each word in a sentence by its position, you can restate this rule as: "There is an S between position p1 and p3, if there is a position p2 such that there is an NP between p1 and p2 and

a VP between p2 and p3." In PROLOG this would be the following axiom, where variables are indicated as atoms with an initial capitalized letter:

```
s(P1, P3) np(P1, P2), vp(P2, P3)
```

To set up the process, add axioms listing the words in the sentence by their position. For example, the sentence *John ate the cat* is described by

```
word(John, 1, 2)
```

```
word(ate, 2, 3)
```

```
word(the, 3, 4)
```

```
word(cat, 4, 5)
```

The lexicon is defined by a set of predicates such as the following:

```
isart(the)
```

```
isname(john)
```

```
isart(the)
```

```
isverb(ate)
```

```
isnoun(cat)
```

Ambiguous words would produce multiple assertions—one for each syntactic category to which they belong.

For each syntactic category, you can define a predicate that is true only if the word between the two specified positions is of that category, as follows:

```
n(I, O) :- word(Word, I, O), isnoun( Word)
```

```
art(I, O) :- word(Word, 1, O), isart(Word)
```

```
v(I, O) :- word(Word, 1, O), isverb(Word)
```

```
n(I, O) :- word(Word, I, O), isnoun( Word)

name(I, O) :- word(Word, I, O), isname( Word)
```

Using the axioms in Figure 3.29, you can prove that *John ate the cat* is a legal sentence by proving $s(1, 5)$, as in Figure 3.30. In Figure 3.30, when there is a possibility of confusing different variables that have the same name, a prime (' \prime) is appended to the variable name to make it unique. This proof trace is in the same format as the trace for the top-down CFG parser, as follows. The state of the proof at any time is the list of subgoals yet to be proven. Since the word positions are included in the goal description, no separate position column need be traced. The backup states are also lists of subgoals, maintained automatically by a system like PROLOG to implement backtracking. A typical trace of a proof in such a system shows only the current state at any time.

Because the standard PROLOG search strategy is the same as the depth-first top-down paring strategy, a parser built from PROLOG will have the same computational complexity, C_i' , that is, the number of steps can be exponential in the

[Allen 1995: Chapter 3 - Grammars and Parsing 74]

```
1. s(P1, P3) :- np(P1, P2), vp(P2, P3)

2. np(P1, P3) :- art(P1, P2), n(P2, P3)

3. np(P1, P3) :- name(P1, P3)

4. pp(P1, P3) :- p(P1, P2), np(P2, P3)

5. vp(P1, P2) :- v(P1, P2)

6. vp(P1, P3) :- v(P1, P2), np(P2, P3)

7. vp(P1, P3) :- v(P1, P2), pp(P2, P3)
```

Figure 3.29 A PROLOG-based representation of Grammar 3.4

Step	Current State	Backup States	Comments
1.	$s(1, 5)$		
2.	$np(1, P2) \ vp(P2, 5)$		
3.	$art(1, P2') \ n(P2', P2)$ $vp(P2, 5)$	$name(1, P2) \ vp(P2, 5)$	fails as no ART at position 1
4.	$name(1, P2) \ vp(P2, 5)$		
5.	$vp(2, 5)$		$name(1, 2)$ proven
6.	$v(2, 5)$	$v(2, P2) \ np(P2, 5)$	fails as no verb spans
		$v(2, P2) \ pp(P2, 5)$	positions 2 to 5
7.	$v(2, P2) \ np(P2, 5)$	$v(2, P2) \ pp(P2, 5)$	

Step	Current State	Backup States	Comments
8.	<code>np(3, 5)</code>	<code>v(2, P2) pp(P2, 5)</code>	<code>v(2, 3)</code> proven
9.	<code>art(3, P2) n(P2, 5)</code>	<code>name(3, 5)</code>	
		<code>v(2, P2) pp(P2, 5)</code>	
10.	<code>n(4, 5)</code>	<code>name(3, 5)</code>	<code>art(3, 4)</code> proven
		<code>v(2, P2) pp(P2, 5)</code>	
11.	<code>^ proof succeeds</code>	<code>name(3, 5)</code>	<code>n(4, 5)</code> proven
		<code>v(2, P2) pp(P2, 5)</code>	

Figure 3.34 A trace of a PROLOG-based parse of *John ate the cat*

length of the input. Even with this worst-case analysis, PROLOG-based grammars can be quite efficient in practice. It is also possible to insert chart-like mechanisms to improve the efficiency of a grammar, although then the simple correspondence between context-free rules and PROLOG rules is lost. Some of these issues will be discussed in the next chapter.

It is worthwhile to try some simple grammars written in PROLOG to better understand top-down, depth-first search. By turning on the tracing facility, you can obtain a trace similar in content to that shown in Figure 3.30.

>> [back](#)

[Allen 1995: Chapter 3 - Grammars and Parsing 75]

Summary

The two basic grammatical formalisms are context-free grammars (CFGs) and recursive transition networks (RTNs). A variety of parsing algorithms can be used for each. For instance, a simple top-down backtracking algorithm can be used for both formalisms and, in fact, the same algorithm can be used in the standard logic-programming-based grammars as well. The most efficient parsers use a chart-like structure to record every constituent built during a parse. By reusing this information later in the search, considerable work can be saved.

>> [back](#)

Related Work and Further Readings

There is a vast literature on syntactic formalisms and parsing algorithms. The notion of context-free grammars was introduced by Chomsky (1956) and has been studied extensively since in linguistics and in computer science. Some of this work will be discussed in detail later, as it is more relevant to the material in the following chapters.

Most of the parsing algorithms were developed in the mid-1960s in computer science, usually with the goal of analyzing programming languages rather than natural language. A classic reference for work in this area is Aho, Sethi, and Ullman (1986), or Aho and Ullman (1972), if the former is not available. The notion of a chart is described in Kay (1973; 1980) and has been adapted by many parsing systems since. The bottom-up chart parser described in this chapter is similar to the left-corner parsing algorithm in Aho and Ullman (1972), while the top-down chart parser is similar to that described by Earley (1970) and hence called the Earley algorithm.

Transition network grammars and parsers are described in Woods (1970; 1973) and parsers based on logic programming are described and compared with transition network systems in Pereira and Warren (1980). Winograd (1983) discusses most of the approaches described here from a slightly different perspective, which could be useful if a specific technique is difficult to understand. Gazdar and Mellish (1989a; 1989b) give detailed descriptions of implementations of parsers in LISP and in PROLOG. In addition, descriptions of transition network parsers can be found in many introductory AI texts, such as Rich and Knight (1992), Winston (1992), and Charniak and McDermott (1985). These books also contain descriptions of the search techniques underlying many of the parsing algorithms. Norvig (1992) is an excellent source on AI programming techniques.

The best sources for work on computational morphology are two books:

Sproat (1992) and Ritchie et al. (1992). Much of the recent work on finite state models has been based on the KIMMO system (Koskenniemi, 1983). Rather than requiring the construction of a huge network, KIMMO uses a set of FSTs which are run in parallel; that is, all of them must simultaneously accept the input and agree on the output. Typically, these FSTs are expressed using an abstract

[Allen 1995: Chapter 3 - Grammars and Parsing 76]

language that allows general morphological rules to be expressed concisely. A compiler can then be used to generate the appropriate networks for the system.

Finite state models are useful for a wide range of processing tasks besides morphological analysis. Blank (1989), for instance, is developing a grammar for English using only finite state methods. Finite state grammars are also used extensively in speech recognition systems.

>> [back](#)

Exercises for Chapter 3

1. (easy)
 - a. Express the following tree in the list notation in Section 3.1.
 - b. Is there a tree structure that could not be expressed as a list structure? How about a list structure that could not be expressed as a tree?
2. (easy) Given the CFG in Grammar 3.4, define an appropriate lexicon and show a trace in the format of Figure 3.5 of a top-down CFG parse of the sentence *The man walked the old dog*.
3. (easy) Given the RTN in Grammar 3.19 and a lexicon in which *green* can be an adjective or a noun, show a trace in the format of Figure 3.21 of a top-down RTN parse of the sentence *The green faded*.
4. (easy) Given the PROLOG-based grammar defined in Figure 3.29, show a trace in the format of Figure 3.30 of the proof that the following is a legal sentence: *The cat ate John*.
5. (medium) Map the following context-free grammar into an equivalent recursive transition network that uses only three networks—an S, NP, and PP network. Make your networks as small as possible.

[Allen 1995: Chapter 3 - Grammars and Parsing 77]

S → **NP VP**

NP2 → **ADJ NP2**

VP → **V**

NP2 → **NP3 PREPS**

VP → **V NP**

NP3 → **N**

VP → **V PP**

PREPS → **PP**

NP → **ART NP2**

PREPS → **PP PREPS**

NP → **NP2**

PP → **NP**

NP2 → **N**

6. (medium) Given the CFG in Exercise 5 and the following lexicon, construct a trace of a pure top-down parse and a pure bottom-up parse of the sentence *The herons fly in groups*. Make your traces as clear as possible, select the rules in the order given in Exercise 5, and indicate all parts of the search. The lexicon entries for each word are

the: ART **heroin:** N

fly: NVADJ

In: P

groups: NV

7. (medium) Consider the following grammar:

S → ADJS N

S → N

ADJS → ADJS ADJ

ADJS → ADJ

Lexicon: ADJ: red, N: house

- a. What happens to the top-down depth-first parser operating on this grammar trying to parse the input *red red*? In particular, state whether the parser succeeds, fails, or never stops.
- b. How about a top-down breadth-first parser operating on the same input *red red*?
- c. How about a top-down breadth-first parser operating on the input *red house*?
- d. How about a bottom-up depth-first parser on *red house*?
- e. For the cases where the parser fails to stop, give a grammar that is equivalent to the one shown in this exercise and that is parsed correctly. (Correct behavior includes failing on unacceptable phrases as well as succeeding on acceptable ones.)
- f. With the new grammar in part (e), do all the preceding parsers now operate correctly on the two phrases *red red* and *red house*?

[Allen 1995: Chapter 3 - Grammars and Parsing 78]

8. (*medium*) Consider the following CFG:

S → NP V

S → NP AUX V

NP → ART N

Trace one of the chart parsers in processing the sentence

1 The 2 man 3 is 4 laughing 5

with the lexicon entries:

the: ART

man: N

is: AUX

laughing: V

Show every step of the parse, giving the parse stack, and drawing the chart each time a nonterminal constituent is added to the chart.

9. (medium) Consider the following CFG that generates sequences of letters:

s → **a** **x** **c**

s → **b** **x** **c**

s → **b** **x** **d**

s → **b** **x** **e**

s → **c** **x** **e**

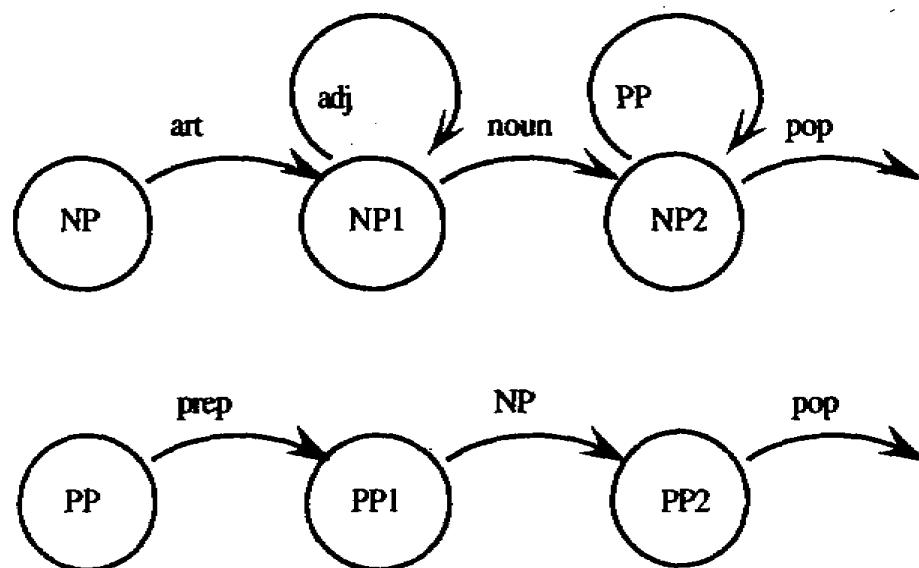
x → **f** **x**

x → **g**

- a. If you had to write a parser for this grammar, would it be better to use a pure top-down or a pure bottom-up approach? Why?

b. Trace the parser of your choice operating on the input *bffge*.

10. (medium) Consider the following CFG and RTN:



[Allen 1995: Chapter 3 - Grammars and Parsing 79]

NP → **ART** **NP1**

NP1 → **ADJ** **N** **PPS**

PPS → **PP**

PPS → **PP** **PPS**

PP → **P** **NP**

- a. State two ways in which the languages described by these two grammars differ. For each, give a sample sentence that is recognized by one grammar but not the other and that demonstrates the difference.
- b. Write a new CFG equivalent to the RTN shown here.
- c. Write a new RTN equivalent to the CFG shown here.

11. (*hard*) Consider the following sentences:

ListA

i. Joe is reading the book.

ii. Joe had won a letter.

iii. Joe has to win.

iv. Joe will have the letter.

v. The letter in the book was read.

vi. The letter must have been in the book by Joe.

vii. The man could have had one.

ListB

i. *Joe has reading the book.

ii. *Joe had win.

iii. *Joe winning.

iv. *Joe will had the letter.

v. *The book was win by Joe.

vi. *Joe will can be mad.

vii. *The man can have having one.

a. Write a context-free grammar that accepts all the sentences in list A while rejecting the sentences in list B. You may find it useful to make reference to the grammatical forms of verbs discussed in Chapter 2.

- b. Implement one of the chart-based parsing strategies and, using the grammar specified in part (a), demonstrate that your parser correctly accepts all the sentences in A and rejects those in B. You should maintain enough information in each entry on the chart so that you can reconstruct the parse tree for each possible interpretation. Make sure your method of recording the structure is well documented and clearly demonstrated.
- c. List three (distinct) grammatical forms that would not be recognized by a parser implementing the grammar in part (a). Provide an example of your own for each of these grammatical forms.

>> [back](#)

Allen 1995 : Natural Language Understanding

	<u>Contents</u>	<u>Preface</u>	<u>Introduction</u>	
<u>previous chapter</u>	<u>Part I Syntactic Processing</u>	<u>Part II Semantic Interpretation</u>	<u>Part III - Context / World Knowledge</u>	<u>next chapter</u>
	<u>Appendices</u>	<u>Bibliography</u>		
	<u>Summaries</u>	<u>Further Readings</u>	<u>Exercises</u>	

Chapter 4: Features and Augmented Grammars

	<u>4.1 Feature Systems and Augmented Grammars</u>
	<u>4.2 Some Basic Feature Systems for English</u>
	<u>4.3 Morphological Analysis and the Lexicon</u>
	<u>4.4 A Simple Grammar Using Features</u>
	<u>4.5 Parsing with Features</u>
	<u>o 4.6 Augmented Transition Networks</u>
	<u>o 4.7 Definite Clause Grammars</u>
	<u>o 4.8 Generalized Feature Systems and Unification Grammars</u>

	<u>4.1 Feature Systems and Augmented Grammars</u>
	<u>Summary</u>
	<u>Related Work and Further Readings</u>
	<u>Exercises for Chapter 4</u>

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 83]

Context-free grammars provide the basis for most of the computational parsing mechanisms developed to date, but as they have been described so far, they would be very inconvenient for capturing natural languages. This chapter describes an extension to the basic context-free mechanism that defines constituents by a set of features. This extension allows aspects of natural language such as agreement and subcategorization to be handled in an intuitive and concise way.

Section 4.1 introduces the notion of feature systems and the generalization of context-free grammars to allow features. Section 4.2 then describes some useful feature systems for English that are typical of those in use in various grammars. Section 4.3 explores some issues in defining the lexicon and shows how using features makes the task considerably simpler. Section 4.4 describes a sample context-free grammar using features and introduces some conventions that simplify the process. Section 4.5 describes how to extend a chart parser to handle a grammar with features. The remaining sections, which are optional, describe how features are used in other grammatical formalisms and explore some more advanced material. Section 4.6 introduces augmented transition networks, which are a generalization of recursive transition networks with features, and Section 4.7 describes definite clause grammars based on PROLOG. Section 4.8 describes generalized feature systems and unification grammars.

>> [back](#)

4.1 Feature Systems and Augmented Grammars

In natural languages there are often agreement restrictions between words and phrases. For example, the NP "*a men*" is not correct English because the article *a* indicates a single object while the noun "*men*" indicates a plural object; the noun phrase does not satisfy the number agreement restriction of English. There are many other forms of agreement, including subject-verb agreement, gender agreement for pronouns, restrictions between the head of a phrase and the form of its complement, and so on. To handle such phenomena conveniently, the grammatical formalism is extended to allow constituents to have features. For example, we might define a feature NUMBER that may take a value of either *s* (for singular) or *p* (for plural), and we then might write an augmented CFG rule such as

NP → ART N only when NUMBER₁ agrees with NUMBER₂

This rule says that a legal noun phrase consists of an article followed by a noun, but only when the number feature of the first word agrees with the number feature of the second. This one rule is equivalent to two CFG rules that would use different terminal symbols for encoding singular and plural forms of all noun phrases, such as

NP-SING → ART-SING N-SING

NP-PLURAL → ART-PLURAL N-PLURAL

While the two approaches seem similar in ease-of-use in this one example, consider that all rules in the grammar that use an NP on the right-hand side would

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 84]

now need to be duplicated to include a rule for NP-SING and a rule for NP-PLURAL, effectively doubling the size of the grammar. And handling additional features, such as person agreement, would double the size of the grammar again and again. Using features, the size of the augmented grammar remains the same as the original one yet accounts for agreement constraints.

To accomplish this, a constituent is defined as a feature structure - a mapping from features to values that defines the relevant properties of the constituent. In the examples in this book, feature names in formulas will be written in boldface. For example, a feature structure for a constituent ART1 that represents a particular use of the word a might be written as follows:

ART1 :

(CAT ART

ROOT a

NUMBER S)

This says it is a constituent in the category ART that has as its root the word a and is singular. Usually an abbreviation is used that gives the CAT value more prominence and provides an intuitive tie back to simple context-free grammars. In this abbreviated form, constituent ART1 would be written as

ART1: (ART ROOT a NUMBER S)

Feature structures can be used to represent larger constituents as well. To do this, feature structures themselves can occur as values. Special features based on the integers - 1, 2, 3, and so on - will stand for the first subconstituent, second subconstituent, and so on, as needed. With this, the representation of the NP constituent for the phrase "*a fish*" could be

NP1: (NP NUMBERs

1 (ART ROOT a

NUMBER s)

2 (N ROOT fish

NUMBER s))

Note that this can also be viewed as a representation of a parse tree shown in Figure 4.1, where the subconstituent features 1 and 2 correspond to the subconstituent links in the tree.

The rules in an augmented grammar are stated in terms of feature structures rather than simple categories. Variables are allowed as feature values so that a rule can apply to a wide range of situations. For example, a rule for simple noun phrases would be as follows:

(NP NUMBER ?n) - (ART NUMBER ?n) (N NUMBER ?n)

This says that an NP constituent can consist of two subconstituents, the first being an ART and the second being an N, in which the NUMBER feature in all three constituents is identical. According to this rule, constituent NP1 given previously is a legal constituent. On the other hand, the constituent

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 85]

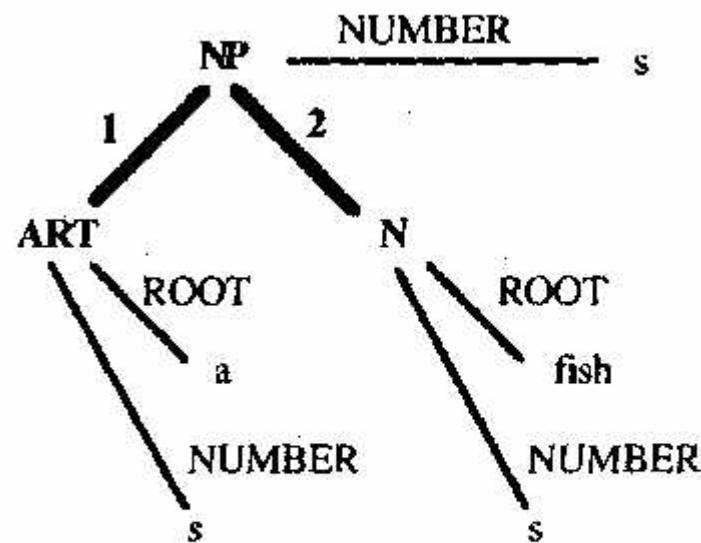


Figure 4.1 Viewing a feature structure as an extended parse tree

Figure 4.1 Viewing a feature structure as an extended parse tree

```
* (NP 1 (ART NUMBER s)  
      2 (N NUMBER s))
```

is not allowed by this rule because there is no NUMBER feature in the NP, and the constituent

```
* (NP NUMBER s  
      1 (ART NUMBER s)  
      2 (N NUMBER p))
```

is not allowed because the NUMBER feature of the N constituent is not identical to the other two NUMBER features.

Variables are also useful in specifying ambiguity in a constituent. For instance, the word fish is ambiguous between a singular and a plural reading. Thus the word might have two entries in the lexicon that differ only by the value of the NUMBER feature. Alternatively, we could define a single entry that uses a variable as the value of the NUMBER feature, that is,

```
(N ROOT fish NUMBER ?n)
```

This works because any value of the NUMBER feature is allowed for the word fish. In many cases, however, not just any value would work, but a range of values is possible.. To handle these cases, we introduce constrained variables, which are variables that can only take a value out of a specified list. For example, the variable ?n{s p} would be a variable that can take the value s or the value p. Typically. when we write such variables, we will drop the variable name altogether and just list the possible values. Given this, the word fish might be represented by the constituent

```
(N ROOT fish NUMBER ?n{sp})
```

or more simply as

```
(N ROOT fish NUMBER {s p})
```

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 86]

BOX 4.1

BOX 4.1 Formalizing Feature Structures

There is an active area of research in the formal properties of feature structures. This work views a feature system as a formal logic. A feature structure is defined as a partial function from features to feature values. For example, the feature structure

ART1: (**CAT ART**
ROOT a
NUMBER s)

is treated as an abbreviation of the following statement in FOPC:

$$ART1(CAT) = ART \wedge ART1(ROOT) = a \wedge ART1(NUMBER) = s$$

Feature structures with disjunctive values map to disjunctions. The structure

THE1: (**CAT ART**
ROOT the
NUMBER {s p})

would be represented as

$$\begin{aligned} THE1(CAT) &= ART \wedge THE1(ROOT) = \text{the} \\ &\wedge (THE1(NUMBER) = s \vee THE1(NUMBER) = p) \end{aligned}$$

Given this, agreement between feature values can be defined as equality equations.

There is an interesting issue of whether an augmented context-free grammar can describe languages that cannot be described by a simple context-free grammar. The answer depends on the constraints on what can be a feature value. If the set of feature values is finite, then it would always be possible to create new constituent categories for every combination of features. Thus it is expressively equivalent to a context-free grammar. If the set of feature values is unconstrained, however, then such grammars have arbitrary computational power. In practice, even when the set of values is not explicitly restricted, this power is not used, and the standard parsing algorithms can be used on grammars that include features.

>> [back](#)

4.2 Some Basic Feature Systems for English

This section describes some basic feature systems that are commonly used in grammars of English and develops the particular set of features used throughout this book. Specifically, it considers number and person agreement, verb form features, and features required to handle subcategorization constraints. You should read this to become familiar with the features in general and then refer back to it later when you need a detailed specification of a particular feature.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 87]

Person and Number Features

In the previous section, you saw the number system in English: Words may be classified as to whether they can describe a single object or multiple objects. While number agreement restrictions occur in several different places

in English, they are most importantly found in subject-verb agreement. But subjects and verbs must also agree on another dimension, namely with respect to the person. The possible values of this dimension are

First Person (1): The noun phrase refers to the speaker, or a group of people including the speaker (for example, I, we, you, and 0).

Second Person (2): The noun phrase refers to the listener, or a group including the listener but not including the speaker (for example, you, all of you).

Third Person (3): The noun phrase refers to one or more objects, not including the speaker or hearer.

Since number and person features always co-occur, it is convenient to combine the two into a single feature, AGR, that has six possible values: first person singular (1s), second person singular (2s), third person singular (3s), and first, second and third person plural (1p, 2p, and 3p, respectively). For example, an instance of the word is can agree only with a third person singular subject, so its AGR feature would be 3s. An instance of the word are, however, may agree with second person singular or any of the plural forms, so its AGR feature would be a variable ranging over the values {2s 1p 2p 3p}.

Verb-Form Features and Verb Subcategorization

Another very important feature system in English involves the form of the verb. This feature is used in many situations, such as the analysis of auxiliaries and generally in the subcategorization restrictions of many head

words. As described in Chapter 2, there are five basic forms of verbs. The feature system for verb forms will be slightly more complicated in order to conveniently capture certain phenomena. In particular, we will use the following feature values for the feature VFORM:

base - base form (for example, go, be, say, decide)

pres - simple present tense (for example, go, goes, am, is, say, says, decide)

past - simple past tense (for example, went, was, said, decided)

fin - finite (that is, a tensed form, equivalent to {pres past})

ing - present participle (for example, going, being, saying, deciding)

pastprt - past participle (for example, gone, been, said, decided)

inf - a special feature value that is used for infinitive forms with the word to

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 88]

Value	Example Verb	Example
_none	laugh	Jack laughed.
_np	find	Jack found a key.
_np_np	give	Jack gave Sue the paper.
_vp:inf	want	Jack wants to fly.
_np_vp:inf	tell	Jack told the man to go.
_vp:ing	keep	Jack keeps hoping for the best.
_np_vp:ing	catch	Jack caught Sam looking at his desk.
_np_vp:base	watch	Jack watched Sam look at his desk.

Figure 4.2 The SUBCAT values for NP/VP combinations

Figure 4.2 The SUBCAT values for NPNP combinations

To handle the interactions between words and their complements, an additional feature, SUBCAT, is used. Chapter 2 described some common verb sub-categorization possibilities. Each one will correspond to a different value of the SUBCAT feature. Figure 4.2 shows some SUBCAT values for complements consisting of combinations of NPs and VPs. To help you remember the meaning of the feature values, they are formed by

listing the main category of each part of the complement. If the category is restricted by a feature value, then the feature value follows the constituent separated by a colon. Thus the value npvp:inf will be used to indicate a complement that consists of an NP followed by a VP with VFORM value inf. Of course, this naming is just a convention to help the reader; you could give these values any arbitrary name, since their significance is determined solely by the grammar rules that involve the feature. For instance, the rule for verbs with a SUBCAT value of _np_vp:inf would be

(VP) → (V **SUBCAT** _np_vp:inf)

(NP)

(VP **VFORM** inf)

This says that a VP can consist of a V with SUBCAT value _np_vp:inf, followed by an NP, followed by a VP with VFORM value inf. Clearly, this rule could be rewritten using any other unique symbol instead of _np_vp:inf, as long as the lexicon is changed to use this new value.

Many verbs have complement structures that require a prepositional phrase with a particular preposition, or one that plays a particular role. For example, the verb give allows a complement consisting of an NP followed by a PP using the preposition to, as in "Jack gave the money to the bank". Other verbs, such as "put", require a prepositional phrase that describes a location, using prepositions such as "in", "inside", "on", and "by". To express this within the feature system, we introduce a feature PFORM on prepositional phrases. A prepositional phrase with a PFORM value such as TO must have the preposition to as its head, and so on. A prepositional phrase with

a PFORM value LOC must describe a location. Another useful PFORM value is MOT, used with verbs such as walk, which may take a

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 89]

Value	Example Prepositions	Example
TO	to	I gave it to the bank.
LOC	in, on, by, inside, on top of	I put it on the desk.
MOT	to, from, along, ...	I walked to the store.

Figure 4.3 Some values of the PFORM feature for prepositional phrases

Figure 4.3 Some values of the PFORM feature for prepositional phrases

Value	Example Verb	Example
_np_pp:to	give	Jack gave the key to the man.
_pp:loc	be	Jack is at the store.
_np_pp:loc	put	Jack put the box in the corner.
_pp:mot	go	Jack went to the store.
_np_pp:mot	take	Jack took the hat to the party.
_adjp	be, seem	Jack is happy.
_np_adjp	keep	Jack kept the dinner hot.
_s:that	believe	Jack believed that the world was flat.
_s:for	hope	Jack hoped for the man to win the prize.

Figure 4.4 Additional SUBCAT values

Figure 4.4 Additional SUBCAT values

prepositional phrase that describes some aspect of a path, as in We walked to the store. Prepositions that can create such phrases include to, from, and along. The LOC and MOT values might seem hard to distinguish, as certain prepositions might describe either a location or a path, but they are distinct. For example, while Jack put the box (in on by] the corner is fine, *Jack put the box (to from along] the corner is ill-formed. Figure 4.3 summarizes the PFOM feature.

This feature can be used to restrict the complement forms for various verbs. Using the naming convention discussed previously, the SUBCAT value of a verb such as put would be jip pp:loc, and the appropriate rule in the grammar would be

(VP) → (V **SUBCAT** _np_pp:loc)

(NP)

(PP **PFOM** LOC)

For embedded sentences, a complementizer is often needed and must be subcategorized for. Thus a COMP feature with possible values for, that, and nocomp will be useful. For example, the verb tell can subcategorize for an S that has the complementizer that. Thus one SUBCAT value of tell will be _s:that. Similarly, the verb wish subcategorizes for an S with the complementizer for, as in We wished for the rain to stop. Thus one value of the

SUBCAT feature for wish is _s:for. Figure 4.4 lists some of these additional SUBCAT values and examples for a variety of verbs. In this section, all the examples with the

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 90]

SUBCAT feature have involved verbs, but nouns, prepositions, and adjectives may also use the SUBCAT feature and subcategorize for their complements in the same way.

Binary Features

Certain features are binary in that a constituent either has or doesn't have the feature. In our formalization a binary feature is simply a feature whose value is restricted to be either + or -. For example, the INV feature is a binary feature that indicates whether or not an S structure has an inverted subject (as in a yes/no question). The S structure for the sentence Jack laughed will have an INV value —, whereas the S structure for the sentence Did Jack laugh? will have the INV value +. Often, the value is used as a prefix, and we would say that a structure has

the feature +INV or -INV. Other binary features will be introduced as necessary throughout the development of the grammars.

The Default Value for Features

It will be useful on many occasions to allow a default value for features. Anytime a constituent is constructed that could have a feature, but a value is not specified, the feature takes the default value of -. This is especially useful for binary features but is used for nonbinary features as well; this usually ensures that any later agreement check on the feature will fail. The default value is inserted when the constituent is first constructed.

>> [back](#)

4.3 Morphological Analysis and the Lexicon

Before you can specify a grammar, you must define the lexicon. This section explores some issues in lexicon design and the need for a morphological analysis component

The lexicon must contain information about all the different words that can be used, including all the relevant feature value restrictions. When a word is ambiguous, it may be described by multiple entries in the lexicon, one for each different use.

Because words tend to follow regular morphological patterns, however, many forms of words need not be explicitly included in the lexicon. Most English verbs, for example, use the same set of suffixes to indicate different forms: -s is added for third person singular present tense, -ed for past tense, -ing for the present participle, and so on. Without any morphological analysis, the lexicon would have to contain every one of these forms. For the verb want this would require six entries, for want (both in base and present form), wants, wanting, and wanted (both in past and past participle form).

In contrast, by using the methods described in Section 3.7 to strip suffixes there needs to be only one entry for want. The idea is to store the base form of the

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 91]

verb in the lexicon and use context-free rules to combine verbs with suffixes to derive the other entries. Consider the following rule for present tense verbs:

```
(V ROOT ?r SUBCAT ?s VFORM pres AGR 3s) ->  
(V ROOT ?r SUBCAT ?s VFORM base) (+S)
```

where +S is a new lexical category that contains only the suffix morpheme -s. This rule, coupled with the lexicon entry

want:

```
(V ROOT want  
SUBCAT {_np_vp:inf _np_vp:inf}  
VFORM base)
```

would produce the following constituent given the input string want -s

want:

```
(V ROOT want  
SUBCAT {_np_vp:inf _np_vp:inf}  
VFORM pres  
AGR 3s)
```

Another rule would generate the constituents for the present tense form not in third person singular, which for most verbs is identical to the root form:

```
(V ROOT ?r SUBCAT ?s VFORM pres AGR {1s 2s 1p 2p 3p})  
→ (V ROOT ?r SUBCAT ?s VFORM base)
```

But this rule needs to be modified in order to avoid generating erroneous interpretations. Currently, it can transform any base form verb into a present tense form, which is clearly wrong for some irregular verbs. For instance, the base form be cannot be used as a present form (for example, *We be at the store). To cover these cases, a feature is introduced to identify irregular forms. Specifically, verbs with the binary feature +IRREG-PRES have irregular present tense forms. Now the rule above can be stated correctly:

```
(V ROOT ?r SUBCAT ?s VFORM pres AGR (1s 2s 1p 2p 3p))  
→ (V ROOT ?r SUBCAT ?s VFORM base IRREG-PRES -)
```

Because of the default mechanism, the IRREG-PRES feature need only be specified on the irregular verbs. The regular verbs default to -, as desired. Similar binary features would be needed to flag irregular past forms (IRREG-PAST, such as saw), and to distinguish -en past participles from -ed past participles (EN-PASTPRT). These features restrict the application of the standard lexical rules, and the irregular forms are added explicitly to the lexicon. Grammar 4.5 gives a set of rules for deriving different verb and noun forms using these features.

Given a large set of features, the task of writing lexical entries appears very difficult. Most frameworks allow some mechanisms that help alleviate these problems. The first technique - allowing default values for features - has already been mentioned. With this capability, if an entry takes a default value for a given feature, then it need not be explicitly stated. Another commonly used technique is

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 92]

Present Tense

1. (V ROOT ?r SUBCAT ?s VFORM pres AGR 3s) →
(V ROOT ?r SUBCAT ?s VFORM base IRREG-PRES →)+S
2. (V ROOT ?r SUBCAT ?s VFORM pres AGR {1s 2s 1p 2p 3p}) →
(V ROOT ?r SUBCAT ?s VFORM base IRREG-PRES →)

Past Tense

3. (V ROOT ?r SUBCAT ?s VFORM past AGR {1s 2s 3s 1p 2p 3p}) →
(V ROOT ?r SUBCAT ?s VFORM base IRREG-PAST →)+ED

Past Participle

4. (V ROOT ?r SUBCAT ?s VFORM pastprt) →
(V ROOT ?r SUBCAT ?s VFORM base EN-PASTPRT →)+ED
5. (V ROOT ?r SUBCAT ?s VFORM pastprt) →
(V ROOT ?r SUBCAT ?s VFORM base EN-PASTPRT +)+EN.

Present Participle

6. (V ROOT ?r SUBCAT ?s VFORM ing) →
(V ROOT ?r SUBCAT ?s VFORM base)+ING

Plural Nouns

7. (N ROOT ?r AGR 3p) →
(N ROOT ?r AGR 3s IRREG-PL →)+S

Grammar 4.5 Some lexical rules for common suffixes on verbs and nouns

Grammar 4.5 Some lexical rules for common suffixes on verbs and nouns

to allow the lexicon writing to define clusters of features, and then indicate a cluster with a single symbol rather than listing them all. Later, additional techniques will be discussed that allow the inheritance of features in a feature hierarchy.

Figure 4.6 contains a small lexicon. It contains many of the words to be used in the examples that follow. It contains three entries for the word "saw" - as a noun, as a regular verb, and as the irregular past tense form of the verb "see" - as illustrated in the sentences

The saw was broken.

Jack wanted me to saw the board in half.

I saw Jack eat the pizza.

With an algorithm for stripping the suffixes and regularizing the spelling, as described in Section 3.7, the derived entries can be generated using any of the basic parsing algorithms on Grammar 4.5. With the lexicon in Figure 4.6 and Grammar 4.5, correct constituents for the following words can be derived: been, being, cries, cried, crying, dogs, saws (two interpretations), sawed, sawing, seen, seeing, seeds, wants, wanting, and wanted. For example,

the word cries would be transformed into the sequence cry +s, and then rule 1 would produce the present tense entry from the base form in the lexicon.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 93]

a:	(CAT ART ROOT A1 AGR 3s)	saw:	(CAT N ROOT SAW1 AGR 3s)
be:	(CAT V ROOT BE1 VFORM base IRREG-PRES + IRREG-PAST + SUBCAT {_adjp _np})	saw:	(CAT V ROOT SAW2 VFORM base SUBCAT _np)
cry:	(CAT V ROOT CRY1 VFORM base SUBCAT _none)	saw:	(CAT V ROOT SEE1 VFORM past SUBCAT _np)
dog:	(CAT N ROOT DOG1 AGR 3s)	see:	(CAT V ROOT SEE1 VFORM base SUBCAT _np IRREG-PAST + EN-PASTPRT +)
fish:	(CAT N ROOT FISH1 AGR {3s 3p} IRREG-PL +)	seed:	(CAT N ROOT SEED1 AGR 3s)
happy:	(CAT ADJ SUBCAT _vp:inf)	the:	(CAT ART ROOT THE1 AGR {3s 3p})
he:	(CAT PRO ROOT HE1 AGR 3s)	to:	(CAT TO)
is:	(CAT V ROOT BE1 VFORM pres SUBCAT {_adjp _np} AGR 3s)	want:	(CAT V ROOT WANT1 VFORM base SUBCAT {_np _vp:inf _np _vp:inf})
Jack:	(CAT NAME AGR 3s)	was:	(CAT V ROOT BE1 VFORM past AGR {1s 3s} SUBCAT {_adjp _np})
man:	(CAT N1 ROOT MAN1 AGR 3s)	were:	(CAT V ROOT BE VFORM past AGR {2s 1p 2p 3p} SUBCAT {_adjp _np})
men:	(CAT N ROOT MAN1 AGR 3p)		

Figure 4.6 A lexicon

Figure 4.6 A lexicon

Often a word will have multiple interpretations that use different entries and different lexical rules. The word saws, for instance, transformed into the sequence saw +s, can be a plural noun (via rule 7 and the first entry for saw), or the third person present form of the verb saw (via rule 1 and the second entry for saw). Note that rule I cannot apply to the third entry, as its VFORM is not base.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 94]

The success of this approach depends on being able to prohibit erroneous derivations, such as analyzing seed as the past tense of the verb "see". This analysis will never be considered if the FST that strips suffixes is correctly designed. Specifically, the word see will not allow a transition to the states that allow the -ed suffix. But even if this were produced for some reason, the IRREG-PAST value + in the entry for see would prohibit rule 3 from applying.

>> [back](#)

4.4 A Simple Grammar Using Features

This section presents a simple grammar using the feature systems and lexicon developed in the earlier sections. It will handle sentences such as the following:

The man cries.

The men cry.

The man saw the dogs.

He wants the dog.

He wants to be happy.

He wants the man to see the dog.

He is happy to be a dog.

It does not find the following acceptable:

*** The men cries.**

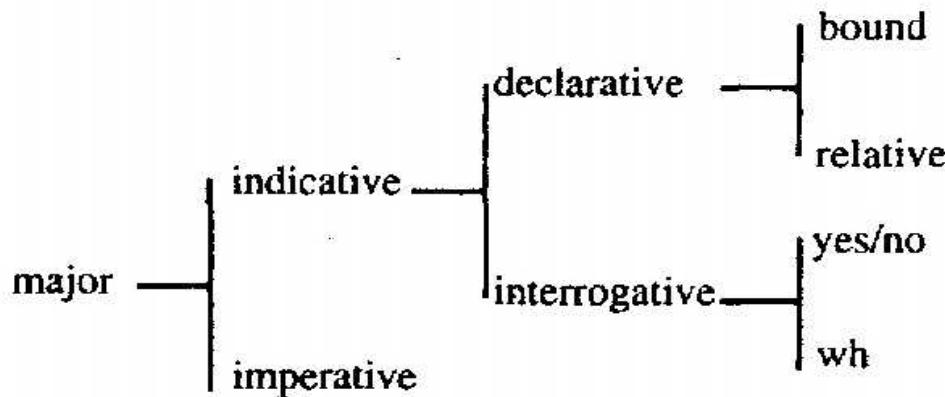
- * **The man cry.**
- * **The man saw to be happy.**
- * **He wants.**
- * **He wants the man saw the dog.**

Before developing the grammar, some additional conventions are introduced that will be very useful throughout the book. It is very cumbersome to write grammatical rules that include all the necessary features. But there are certain regularities in the use of features that can be exploited to simplify the process of writing ‘rules’. For instance, many feature values are unique to a feature (for example, the value inf can only appear in the VFORM feature, and _np_vp:inf can only appear in the SUBCAT feature). Because of this, we can omit the feature name without introducing any ambiguity. Unique feature values will be listed using square parentheses. Thus (VP SUBCAT inf) will be abbreviated as VP[inf]. Since binary features do not have unique values, a special convention is introduced for them. For a binary feature B, the constituent C[+B] indicates the constituent (C B +).

Many features are constrained so that the value on the mother must be identical to the value on its head subconstituent. These are called head features. For instance, in all VP rules the VFORM and AGR values are the same in the VP and the head verb, as in the rule

BOX 4.2 Systemic Grammar

An important influence on the development of computational feature-based systems was systemic grammar (Halliday, 1985). This theory emphasizes the functional role of linguistic constructs as they affect communication. The grammar is organized as a set of choices about discourse function that determine the structure of the sentence. The choices are organized into hierarchical structures called systems. For example, the mood system would capture all the choices that affect the mood of the sentence. Part of this structure looks as follows:



This structure indicates that once certain choices are made, others become relevant. For instance, if you decide that a sentence is in the declarative mood, then the choice between bound and relative becomes relevant. The choice between yes/no and wh, on the other hand, is not relevant to a declarative sentence.

Systemic grammar was used in Winograd (1973), and Winograd (1983) contains a good discussion of the formalism. In recent years it has mainly been used in natural language generation systems because it provides a good formalism for organizing the choices that need to be made while planning a sentence (for example, see Mann and Mathiesson (1985) and Patten (1988)).

```
(VP VFORM ?v AGR ?a) ->  
(V VFORM ?v AGR ?a SUBCAT _np_vp:inf)  
(NP)  
(VP VFORM inf)
```

If the head features can be declared separately from the rules, the system can automatically add these features to the rules as needed. With VFORM and AGR declared as head features, the previous VP rule can be abbreviated as

```
VP -> (V SUBCAT _np_vp:inf) NP (VP VFORM inf)
```

The head constituent in a rule will be indicated in italics. Combining all the abbreviation conventions, the rule could be further simplified to

VP → V [*_vp_vp*:inf] NP VP[inf]

A simple grammar using these conventions is shown as Grammar 4.7. Except for rules 1 and 2, which must enforce number agreement, all the rest of the feature constraints can be captured using the conventions that have been

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 96]

1. $S[-\text{inv}] \rightarrow (\text{NP } AGR \ ?a) (\text{VP}[\{\text{pres past}\}] AGR \ ?a)$
2. $\text{NP} \rightarrow (\text{ART } AGR \ ?a) (N AGR \ ?a)$
3. $\text{NP} \rightarrow PRO$
4. $\text{VP} \rightarrow V[_{\text{none}}]$
5. $\text{VP} \rightarrow V[_{\text{np}}] \text{NP}$
6. $\text{VP} \rightarrow V[_{\text{vp:inf}}] \text{VP[inf]}$
7. $\text{VP} \rightarrow V[_{\text{np_vp:inf}}] \text{NP VP[inf]}$
8. $\text{VP} \rightarrow V[_{\text{adjp}}] \text{ADJP}$
9. $\text{VP[inf]} \rightarrow TO \text{ VP[base]}$
10. $\text{ADJP} \rightarrow ADJ$
11. $\text{ADJP} \rightarrow ADJ[_{\text{vp:inf}}] \text{VP[inf]}$

Head features for S, VP: VFORM, AGR

Head features for NP: AGR

Grammar 4.7 A simple grammar in abbreviated form

Grammar 4.7 A simple grammar in the abbrveiated form

1. $(S \text{ INV} - VFORM ?v \{\text{pres past}\} AGR ?a) \rightarrow (NP AGR ?a) (VP VFORM ?v \{\text{pres past}\} AGR ?a)$
2. $(NP AGR ?a) \rightarrow (ART AGR ?a) (NAGR ?a)$
3. $(NP AGR ?a) \rightarrow (PRO AGR ?a)$
4. $(VP AGR ?a VFORM ?v) \rightarrow (V SUBCAT_none AGR ?a VFORM ?v)$
5. $(VP AGR ?a VFORM ?v) \rightarrow (V SUBCAT_np AGR ?a VFORM ?v) NP$
6. $(VP AGR ?a VFORM ?v) \rightarrow (V SUBCAT_vp:inf AGR ?a VFORM ?v) (VP VFORM inf)$
7. $(VP AGR ?a VFORM ?v) \rightarrow (V SUBCAT_np_vp:inf AGR ?a VFORM ?v) NP (VP VFORM inf)$
8. $(VP AGR ?a VFORM ?v) \rightarrow (V SUBCAT_adjp AGR ?a VFORM ?v) ADJP$
9. $(VP SUBCAT inf AGR ?a VFORM inf) \rightarrow (TO AGR ?a VFORM inf) (VP VFORM base)$
10. $ADJP \rightarrow ADJ$
11. $ADJP \rightarrow (ADJ SUBCAT_inf) (VP VFORM inf)$

Grammar 4.8 The expanded grammar showing all features

Grammar 4.8 The expanded grammar showing all features

introduced. The head features for each category are declared at the bottom of the figure. This grammar is an abbreviation of Grammar 4.8. Consider how rule 1 in Grammar 4.7 abbreviates rule 1 in Grammar 4.8. The abbreviated rule is

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 97]

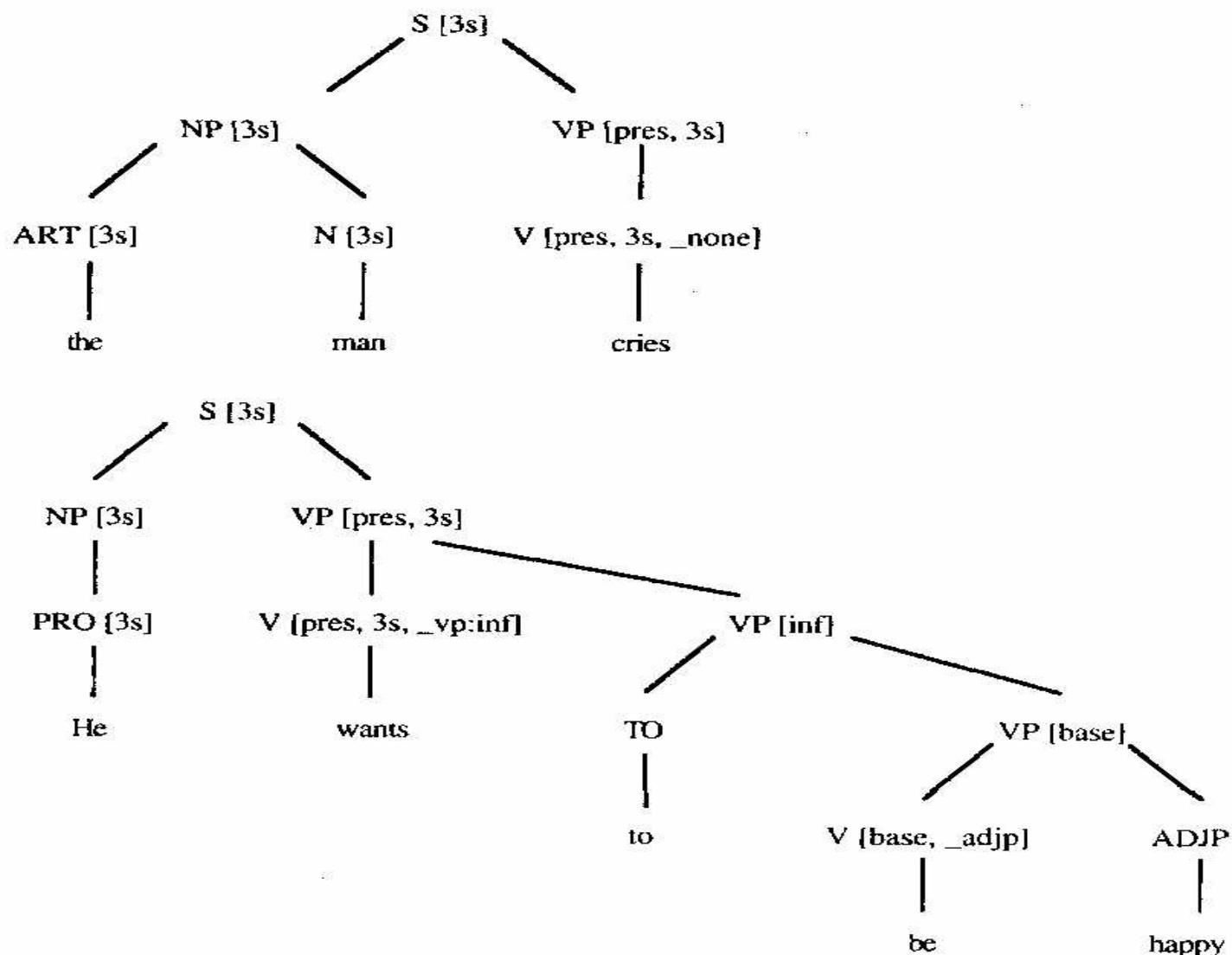


Figure 4.9 Two sample parse trees with feature values

Figure 4.9 Two sample parse trees with feature values

$S[-\text{inv}] \rightarrow (\text{NP } \mathbf{AGR} \ ?a) \ (\text{VP } [\{\text{pres past}\}] \ \mathbf{AGR} \ ?a)$

The unique values can be expanded in the obvious way: the value [-inv] becomes (INV -) and the value [{pres past}] becomes (VFORM ?v{pres past}). The head features for S are AGR and VFORM, so these features must be added to the S and VP head. The resulting rule is

$(S \ \text{INV} = \mathbf{VFORM} \ ?v\{\text{pres past}\} \ \mathbf{AGR} \ ?a) \rightarrow$

$(\text{NP } \mathbf{AGR} \ ?a)$

$(\text{VP } \mathbf{VFORM} \ ?v \ \{\text{pres past}\} \ \mathbf{AGR} \ ?a)$

as shown in Grammar 4.8.

The abbreviated form is also very useful for summarizing parse trees. For instance, Figure 4.9 shows the parse trees for two of the previous sample sentences, demonstrating that each is an acceptable sentence.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 98]

Consider why each of the ill-formed sentences introduced at the beginning of this section are not accepted by Grammar 4.7. Both *"The men cries" and *"The man cry" are not acceptable because the number agreement restriction on rule 1 is not satisfied: The NP constituent for the men has the AGR value 3p, while the VP cries has the AGR value 3s. Thus rule 1 cannot apply. Similarly, "the man cry" is not accepted by the grammar since the man has AGR 3s and the VP cry has as its AGR value a variable ranging over (1s 2s 1p 2p 3p). The phrase *"the man saw to be happy" is not accepted because the verb saw has a SUBCAT value _np. Thus only rule 5 could be used to build a VP. But rule 5 requires an NP complement, and it is not possible for the words "to be happy" to be a legal NP.

The phrase "He wants" is not accepted since the verb wants has a SUBCAT value ranging over {_np_vp:inf _np_vp:inf}, and thus only rules 5, 6, and 7 could apply to build a VP. But all these rules require a nonempty complement of some kind. The phrase *"He wants the man saw the dog" is not accepted for similar reasons, but this requires a little more analysis. Again, rules 5, 6, and 7 are possible with the verb wants. Rules 5 and 6 will not work, but rule 7 looks close, as it requires an NP and a VP[inf]. The phrase "the man" gives us the required NP, but "saw the dog" fails to be a VP[inf]. In particular, "saw the man" is a legal VP, but its VFORM feature will be past, not inf.

>> [back](#)

4.5 Parsing with Features

The parsing algorithms developed in Chapter 3 for context-free grammars can be extended to handle augmented context-free grammars. This involves generalizing the algorithm for matching rules to constituents. For instance, the chart-parsing algorithms developed in Chapter 3 all used an operation for extending active arcs with a new constituent. A constituent X could extend an arc of the form

$C \rightarrow C_1 \dots C_i \circ X \dots C_n$

to produce a new arc of the form

$C \rightarrow C_1 \dots C_i X \circ \dots C_n$

A similar operation can be used for grammars with features, but the parser may have to instantiate variables in the original arc before it can be extended by X. The key to defining this matching operation precisely is to remember the definition of grammar rules with features. A rule such as

1. $(NP \text{ } \mathbf{AGR} \text{ } ?a) \rightarrow \circ (\mathbf{ART} \text{ } \mathbf{AGR} \text{ } ?a) \text{ } (N \text{ } \mathbf{AGR} \text{ } ?a)$

says that an NP can be constructed out of an ART and an N if all three agree on the AGR feature. It does not place any restrictions on any other features that the NP, ART, or N may have. Thus, when matching constituents against

this rule, the only thing that matters is the AGR feature. All other features in the constituent can be ignored. For instance, consider extending arc 1 with the constituent

2. (**ART ROOT** A **AGR** 3s)

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 99]

To make arc 1 applicable, the variable ?a must be instantiated to 3s, producing

3. (NP **AGR** 3s) \rightarrow o (**ART AGR** 3s) (N **AGR** 3s)

This arc can now be extended because every feature in the rule is in constituent 2:

4. (NP **AGR** 3s) \rightarrow (**ART AGR** 3s) o (N **AGR** 3s)

Now, consider extending this arc with the constituent for the word dog:

5. (N **ROOT** DOG1 **AGR** 3s)

This can be done because the AGR features agree. This completes the arc

6. (NP **AGR** 3s) \rightarrow (**ART AGR** 3s) (N **AGR** 3s)

This means the parser has found a constituent of the form (NP AGR 3s).

This algorithm can be specified more precisely as follows: Given an arc A, where the constituent following the dot is called NEXT, and a new constituent X, which is being used to extend the arc,

- a. Find an instantiation of the variables such that all the features specified in NEXT are found in X.
- b. Create a new arc A', which is a copy of A except for the instantiations of the variables determined in step (a).
- c. Update A' as usual in a chart parser.

For instance, let A be arc 1, and X be the ART constituent 2. Then NEXT will be (ART AGR ?a). In step a, NEXT is matched against X, and you find that ?a must be instantiated to 3s. In step b, a new copy of A is made, which is shown as arc 3. In step c, the arc is updated to produce the new arc shown as arc 4.

When constrained variables, such as ?a{3s 3p}, are involved, the matching proceeds in the same manner, but the variable binding must be one of the listed values. If a variable is used in a constituent, then one of its possible values must match the requirement in the rule. If both the rule and the constituent contain variables, the result is a variable ranging over the intersection of their allowed values. For instance, consider extending arc 1 with the constituent (ART ROOT the AGR ?v{3s 3p}), that is, the word "the". To apply, the variable ?a would have to be instantiated to ?v{ 3s 3p}, producing the rule

(NP **AGR** ?v{ 3s 3p}) \rightarrow (**ART AGR** ?v{ 3s 3p}) \circ (N **AGR** ?v{ 3s 3p})

This arc could be extended by (N ROOT dog AGR 3s), because $?v\{3s\ 3p\}$ could be instantiated by the value 3s. The resulting arc would be identical to arc 6. The entry in the chart for the is not changed by this operation. It still has the value $?v\{3s\ 3p\}$. The AGR feature is restricted to 3s only in the arc.

Another extension is useful for recording the structure of the parse. Subconstituent features (1, 2, and so on, depending on which subconstituent is being added) are automatically inserted by the parser each time an arc is extended. The values of these features name subconstituents already in the chart.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 100]

<p>S1 CAT S AGR 3s VFORM pres INV- 1 NP1 2 VP3</p>			
<p>VP3 CAT VP VFORM pres AGR 3s 1 V1 2 VP2</p>			
<p>VP2 CAT VP VFORM inf 1 TO1 2 VP1</p>			
<p>NP1 CAT NP AGR 3s 1 PRO1</p>			<p>VP1 CAT VP VFORM base 1 V2</p>
<p>PRO1 CAT PRO AGR 3s</p>	<p>V1 CAT V ROOT want VFORM pres AGR 3s SUBCAT {<u>_np_vp:inf,</u> <u>_np_vp:inf}</u>}</p>	<p>TO1 CAT TO</p>	<p>V2 CAT V ROOT cry VFORM base SUBCAT _none</p>
He	wants	to	cry

Figure 4.10 The chart for *He wants to cry*.

Figure 4.10 The chart for "He wants to cry".

With this treatment, and assuming that the chart already contains two constituents, ARTL and Ni, for the words the and dog, the constituent added to the chart for the phrase the dog would be

(NP **AGR** 3s

1 ART1

2 N1)

where ART1 (ART ROOT the AGR {3s 3p}) and N1 = (N ROOT dog AGR {3s}). Note that the AGR feature of ART1 was not changed. Thus it could be used with other interpretations that require the value 3p if they are possible. Any of the chart-parsing algorithms described in Chapter 3 can now be used with an augmented grammar by using these extensions to extend arcs and build constituents. Consider an example. Figure 4.10 contains the final chart produced from parsing the sentence He wants to cry using Grammar 4.8. The rest of this section considers how some of the nonterminal symbols were constructed for the chart.

Constituent NP1 was constructed by rule 3, repeated here for convenience:

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 101]

3. (NP **AGR** ?a) \rightarrow (PRO **AGR** ?a)

To match the constituent PRO1, the variable ?a must be instantiated to 3s. Thus the new constituent built is

NP1: (**CAT** NP

AGR 3s

1 PRO1)

Next consider constructing constituent VP1 using rule 4, namely

4. (VP **AGR** ?a **VFORM** ?v) \rightarrow (V **SUBCAT** none **AGR** ?a **VFORM** ?v)

For the right-hand side to match constituent V2, the variable ?v must be instantiated to base. The AGR feature of V2 is not defined, so it defaults to -. The new constituent is

VP1: (**CAT** VP

AGR -

VFORM base

1 V2)

Generally, default values are not shown in the chart. In a similar way, constituent VP2 is built from TO1 and VP1 using rule 9, VP3 is built from V1 and VP2 using rule 6, and S1 is built from NP1 and VP3 using rule 1.

>> [back](#)

4.6 Augmented Transition Networks

Features can also be added to a recursive transition network to produce an augmented transition network (ATN). Features in an ATN are traditionally called registers. Constituent structures are created by allowing each network to have a set of registers. Each time a new network is pushed, a new set of registers is created. As the network is traversed, these registers are set to values by actions associated with each arc. When the network is popped, the registers are assembled to form a constituent structure, with the CAT slot being the network name.

Grammar 4.11 is a simple NP network. The actions are listed in the table below the network. ATNs use a special mechanism to extract the result of following an arc. When a lexical arc, such as arc 1, is followed, the constituent built from the word in the input is put into a special variable named "*". The action

DET := *

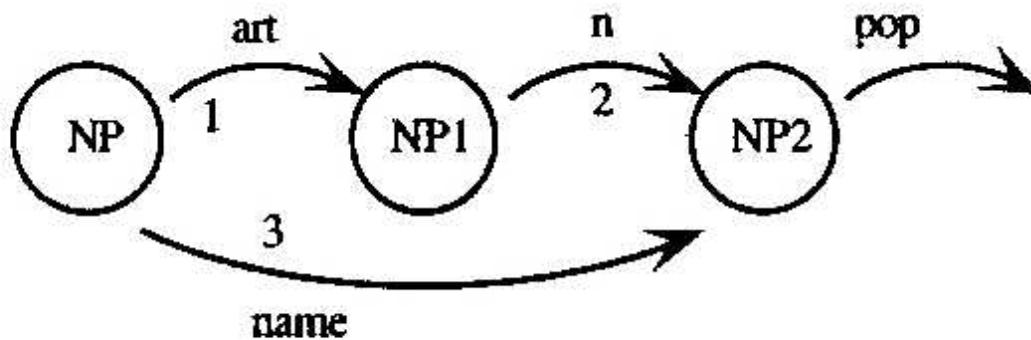
then assigns this constituent to the DET register. The second action on this arc,

AGR := **AGR***

assigns the AGR register of the network to the value of the AGR register of the new word (the constituent in "*").

Agreement checks are specified in the tests. A test is an expression that succeeds if it returns a nonempty value and fails if it returns the empty set or nil.

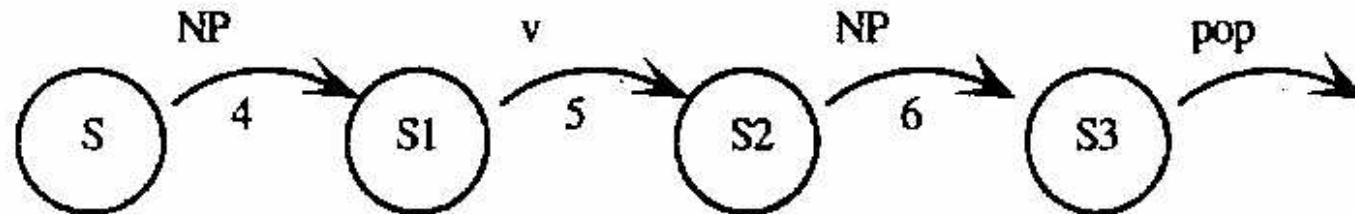
[Allen 1995 : Chapter 4 - Features and Augmented Grammars 102]



Arc	Test	Actions
1	none	DET := * AGR ≈ AGR.
2	AGR ∩ AGR.	HEAD := * AGR ≈ AGR ∩ AGR.
3	none	NAME := * AGR ≈ AGR.

Grammar 4.11 A simple NP network

Grammar 4.11 A simple NP network



Arc	Test	Actions
4	none	SUBJ := *
5	$\text{AGR}_{\text{SUBJ}} \cap \text{AGR}_*$	MAIN-V := * $\text{AGR} = \text{AGR}_{\text{SUBJ}} \cap \text{AGR}_*$
6	none	OBJ := *

Grammar 4.12 A simple S network

Grammar 4.12 A simple S network

If a test fails, its arc is not traversed. The test on arc 2 indicates that the arc can be followed only if the AGR feature of the network has a non-null intersection with the AGR register of the new word (the noun constituent in "*").

Features on push arcs are treated similarly. The constituent built by traversing the NP network is returned as the value "*". Thus in Grammar 4.12, the action on the arc from S to S1,

```
SUBJ := *
```

would assign the constituent returned by the NP network to the register SUBJ. The test on arc 2 will succeed only if the AGR register of the constituent in the SUBJ register has a non-null intersection with the AGR register of the new constituent (the verb). This test enforces subject-verb agreement.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 103]

Trace of S Network

Step	Node	Position	Arc Followed	Registers Set
1.	S	1	arc 4 succeeds (for recursive call see trace below)	SUBJ \leftarrow (NP DET the HEAD dog AGR 3s)
5.	S1	3	arc 5 (checks if 3p \cap 3p)	MAIN-V \leftarrow saw AGR \leftarrow 3p
6.	S2	4	arc 6 (for recursive call trace, see below)	OBJ \leftarrow (NP NAME Jack AGR 3s)
9.	S3	5	pop arc succeeds	returns (S SUBJ (NP DET the HEAD dog AGR 3s) MAIN-V saw AGR 3p OBJ (NP NAME Jack AGR 3s))

Trace of First NP Call: Arc 4

Step	Node	Position	Arc Followed	Registers Set
2.	NP	1	1	DET \leftarrow the AGR \leftarrow {3s 3p}
3.	NP1	2	2 (checks if {3s 3p} \cap 3p)	HEAD \leftarrow dog
4.	NP2	3	pop	returns (NP DET the HEAD dog AGR 3s)

Trace of Second NP Call: Arc 6

Step	Node	Position	Arc Followed	Registers Set
7.	NP	4	3	NAME \leftarrow John AGR \leftarrow 3s
8.	NP2	5	pop	returns (NP NAME John AGR 3s)

Figure 4.13 Trace tests and actions used with *1 The 2 dog 3 saw 4 Jack 5*

Figure 4.13 Trace tests and actions used with "₁ The ₂ dog ₃ saw ₄ Jack ₅"

With the lexicon in Section 4.3, the ATN accepts the following sentences:

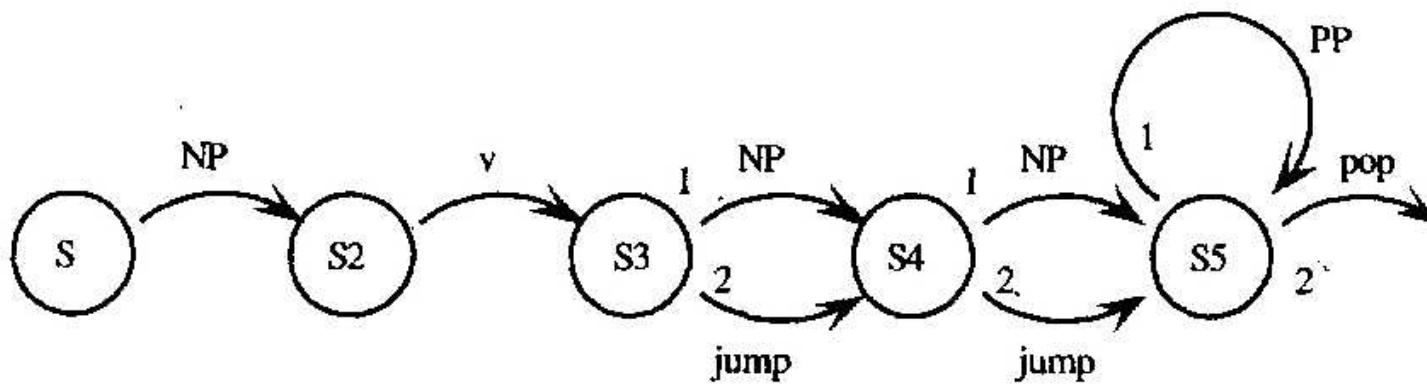
The dog cried.

The dogs saw Jack.

Jack saw the dogs.

Consider an example. A trace of a parse of the sentence "The dog saw Jack" is shown in Figure 4.13. It indicates the current node in the network, the current

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 104]



Arc	Test	Actions
S/1		SUBJ := *
		MOOD := DECL
S2/1	AGR_{SUBJ} ∩ AGR_*	MAIN-V := *
S3/1	SUBCAT_{MAIN-V} ∩ { _np _np _np }	OBJ := *
S3/2	SUBCAT_{MAIN-V} ∩ _none	SUBCAT := SUBCAT_{MAIN-V} ∩ { _np _np _np }
S4/1	SUBCAT_{MAIN-V} ∩ _np_np	SUBCAT := _none IOBJ := OBJ OBJ := *
S5/1	-----	MODS := Append(MODS , *)

Grammar 4.14 An S network for assertions

Grammar 4.14 An S network for assertions

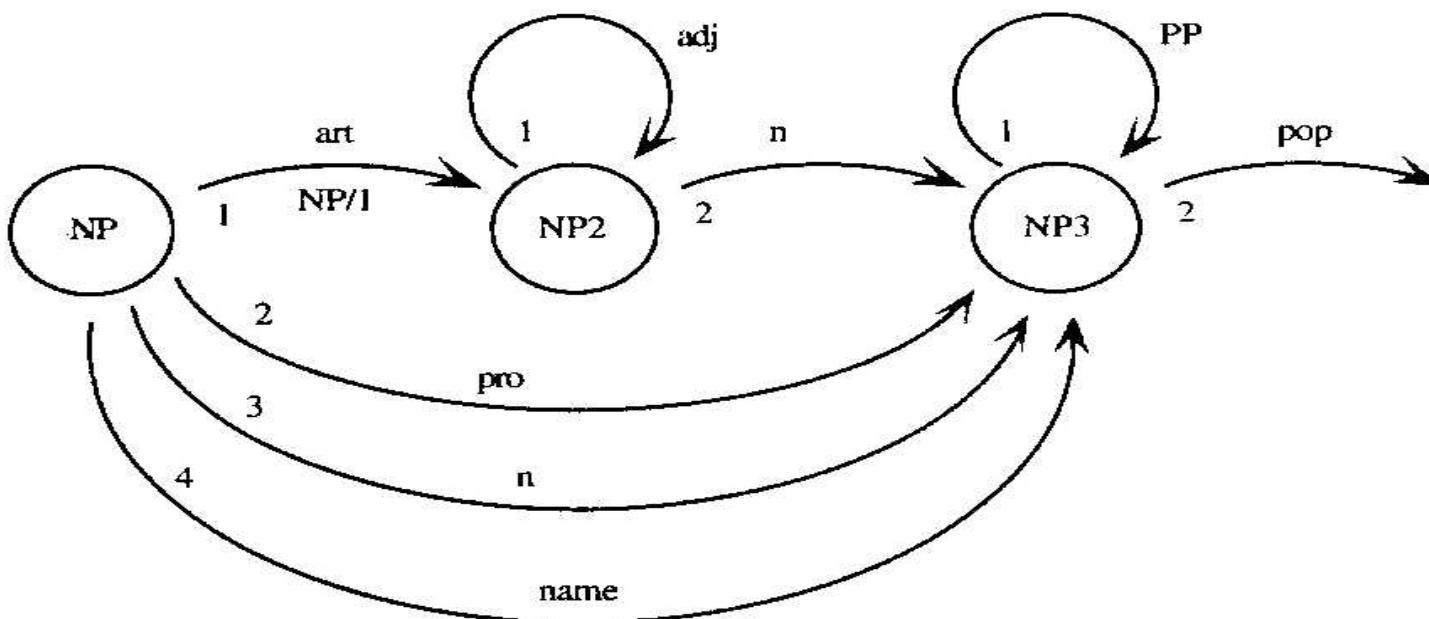
word position, the arc that is followed from the node, and the register manipulations that are performed for the successful parse. It starts in the S network but moves immediately to the NP network from the call on arc 4. The NP network checks for number agreement as it accepts the word sequence The dog. It constructs a noun phrase with the AGR feature plural. When the pop arc is followed, it completes arc 4 in the S network. The NP is assigned to the SUBJ register and then checked for agreement with the verb when arc 3 is followed. The NP "Jack" is accepted in another call to the NP network.

An ATN Grammar for Simple Declarative Sentences

Here is a more comprehensive example of the use of an ATN to describe some declarative sentences. The allowed sentence structure is an initial NP followed by a main verb, which may then be followed by a maximum of two NPs and many PPs, depending on the Verb. Using the feature system extensively, you can create a grammar that accepts any of the preceding complement forms, leaving the actual verb-complement agreement to the feature restrictions. Grammar 4.14 shows the S network. Arcs are numbered using the conventions discussed in Chapter 3. For instance, the arc S3/1 is the arc labeled 1 leaving node 83. The NP network in Grammar 4.15 allows simple

names, bare plural nouns, pronouns, and a simple sequence of a determiner followed by an adjective and a head noun.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 105]



Arc	Test	Actions
NP/1	-----	DET := *
NP/2	-----	PRO := *
NP/3	AGR * \cap 3p	HEAD := * AGR := AGR *
NP/4	-----	NAME := *
NP2/1	-----	ADJS := Append(ADJS , *)
NP2/2	AGR \cap AGR *	HEAD := * AGR := AGR \cap AGR *
NP3/1	-----	MODS := Append(MODS , *)

Grammar 4.15 The NP network

Grammar 4.15 The NP network

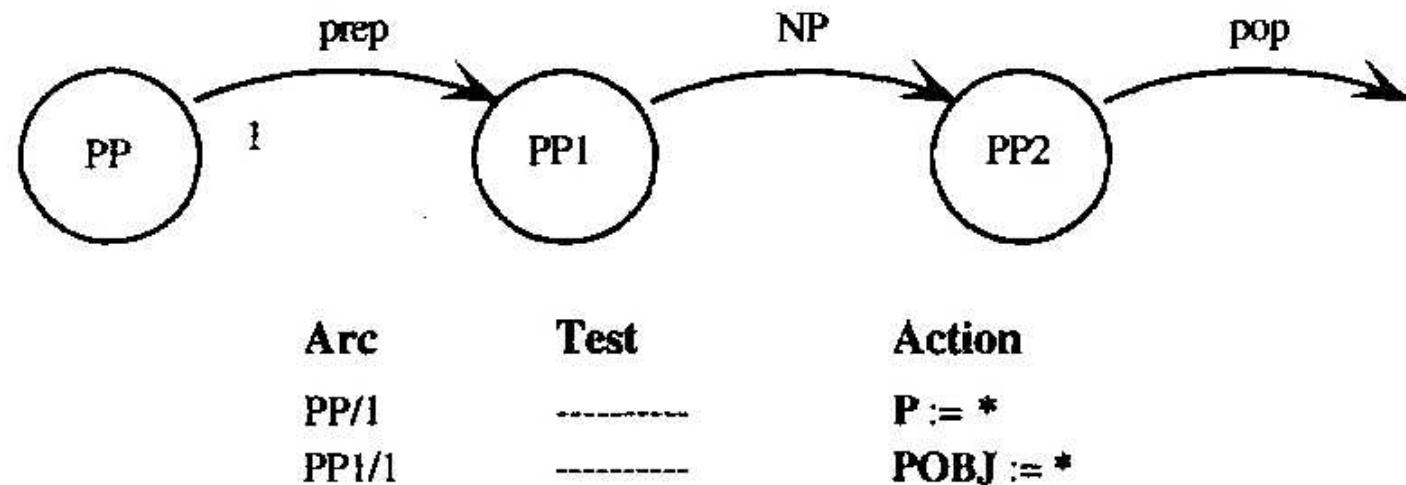
Allowable noun complements include an optional number of prepositional phrases. The prepositional phrase network in Grammar 4.16 is straightforward. Examples of parsing sentences with this grammar are left for the exercises.

Presetting Registers

One further extension to the feature-manipulation facilities in ATNs involves the ability to preset registers in a network as that network is being called, much like parameter passing in a programming language. This facility, called the **SENDR** action in the original ATN systems, is useful to pass information to the network that aids in analyzing the new constituent.

Consider the class of verbs, including want and pray, that accept complements using the infinitive forms of verbs, which are introduced by the word to. According to the classification in Section 4.2, this includes the following:

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 106]



Grammar 4.16 The PP network

Grammar 4.16 The PP network

_vp:inf **Mary wants to have a party.**

_np_vp:inf **Mary wants John to have a party.**

In the context-free grammar developed earlier, such complements were treated as VPs with the VFORM value inf. To capture this same analysis in an ATN, you would need to be able to call a network corresponding to VPs but preset the VFORM register in that network to inf. Another common analysis of these constructs is to view the complements as a special form of sentence with an understood subject. In the first case it is Mary who would be the understood subject (that is, the host), while in the other case it is John. To capture this analysis, many ATN grammars preset the SUBJ register in the new S network when it is called.

>> [back](#)

o 4.7 Definite Clause Grammars

You can augment a logic grammar by adding extra arguments to each predicate to encode features. As a very simple example, you could modify the PROLOG rules to enforce number agreement by adding an extra argument for the number on every predicate for which the number feature is relevant. Thus you would have rules such as those shown in Grammar 4.17.

Consider parsing the noun phrase "the dog cried", which would be captured by the assertions

```
word(the, 1, 2) :-  
word(dog, 2, 3) :-
```

With these axioms, when the word the is parsed by rule 2 in Grammar 4.17, the number feature for the word is returned. You can see this in the following trace of the simple proof of

```
np(1, Number, 3)
```

Using rule 1, you have the following subgoals:

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 107]

1. np(P1, Number, P3) :- art(P1, Number, P2), n(2, Number, P3)
2. art(I, Number, O) :- word(Word, I, O), isart(Word, Number)
3. isart(a, 3s) :-
4. isart(the, 3s) :-
5. isart(the, 3p) :-
6. n(I, Number, O) :- word(Word, I, O), isnoun(Word, Number)
7. isnoun(dog, 3s) :-
8. isnoun(dogs, 3p) :-

Grammar 4.17

Grammar 4.17

1. $s(P1, \text{Number}, s(Np, Vp), P3) :-$
 $np(P1, \text{Number}, Np, P2), vp(P2, \text{Number}, Vp, P3)$
2. $np(P1, \text{Number}, np(Art, N), P3) :-$
 $art(P1, \text{Number1}, Art, P2), n(P2, \text{Number2}, N, P3)$
3. $vp(P1, \text{Number}, vp(Verb), P2) :-$
 $v(P1, Verb, P2)$
4. $art(I, \text{Number}, art(Word), O) :-$
 $word(Word, I, O), isart(Word, Number)$
5. $n(I, \text{Number}, n(Word), O) :-$
 $word(Word, I, O), isnoun(Word, Number)$
6. $v(I, \text{Number}, v(Word), O) :-$
 $word(Word, I, O), isverb(Word, Number)$

Grammar 4.18

Grammar 4.18

`art(I, Number, P2)`

```
n(P2, Number, 3)
```

The first subgoal succeeds by using rule 2 and proving

```
word(the, 1, 2)
```

```
isart(the, 3s)
```

which binds the variables in rule 2 as follows:

```
Number <- 3s
```

```
P2 <- 2
```

Thus, the second subgoal now is

```
n(2, 3s, 3)
```

Using rule 6, this reduces to the subgoals **word**(Word, 2, 3) and **isnoun**(Word, 3s), which are established by the input and rule 7, respectively, with **Word** bound to **dog**. Thus the parse succeeds and the number agreement was enforced.

The grammar can also be extended to record the structure of the parse by adding another argument to the rules. For example, to construct a parse tree, you could use the rules shown in Grammar 4.18. These rules would allow you to prove the following on the sentence "The dog cried":

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 108]

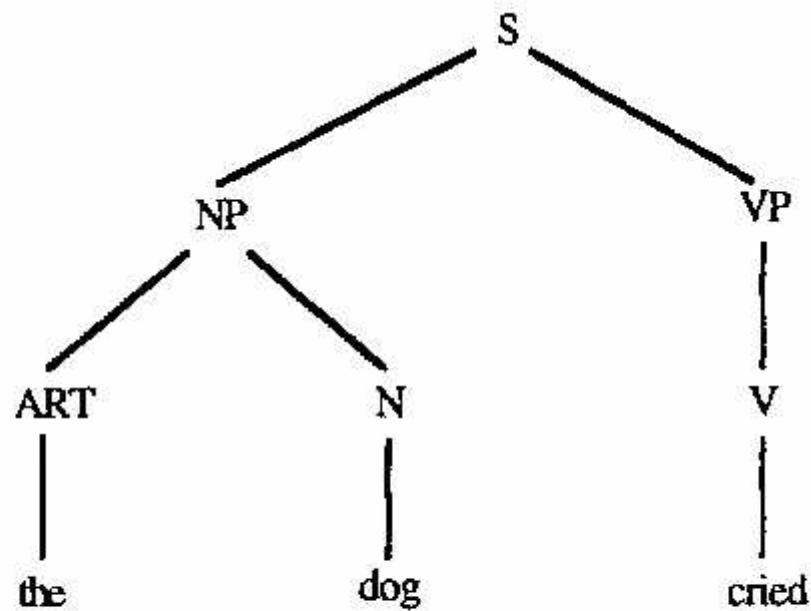


Figure 4.19 A tree representation of the structure

Figure 4.19 A tree representation of the structure

```
s(1, 3s, s(np(art(the), n(dog)), vp(v(cried))), 4)
```

In other words, between positions 1 and 4 there is a sentence with number feature 3s and the structure

```
s(np(art(the), n(dog)), vp(v(cried)))
```

which is a representation of the parse tree shown in Figure 4.19.

For specifying grammars, most logic-based grammar systems provide a more convenient format that is automatically converted into PROLOG clauses like those in Grammar 4.18. Since the word position arguments are on every predicate, they can be omitted and inserted by the system. Similarly, since all the predicates representing terminal symbols (for example, art, N, V) are defined systematically, the system can generate these rules automatically. These abbreviations are encoded in a format called definite clause grammars (DCGs).

For example, you could write Grammar 4.18 by specifying just three DCG rules. (The symbol " \rightarrow " is traditionally used to signal DCG rules.)

```
s (Number, s(Np, Vp)) -> np(N1, Np), vp(N2, Vp)
```

```
np (Number, np(Art, N)) -> art(N1, Art), n(N2, N)
```

```
vp (Number, vp(Verb)) -> v(Number, Verb)
```

The result is a formalism similar to the augmented context-free grammars described in Section 4.3. This similarity can be made even closer if a feature/value representation is used to represent structure. For instance, let us replace all the argument positions with a single argument that is a feature structure. The DCG equivalent to the first five

rules in Grammar 4.7 would be as shown in Grammar 4.20, where square parentheses indicate lists in PROLOG. Notice that rule 1 will not succeed unless the S, NP, and VP all agree on the agr feature, because they have the same variable as a value.

Of course, this encoding will only work if every constituent specifies its feature values in the same order so that the feature lists unify. For example, if the

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 109]

1. $s([\text{inv} - \text{agr} \text{ Agr}]) \rightarrow np([\text{agr} \text{ Agr}]), vp([\text{agr} \text{ Agr} \text{ vform pres}])$
2. $np([\text{agr} \text{ Agr2}]) \rightarrow \text{art}([\text{agr} \text{ Agr2}]), n([\text{agr} \text{ Agr2}])$
3. $np([\text{agr} \text{ Agr3}]) \rightarrow \text{pro}([\text{agr} \text{ Agr3}])$
4. $vp([\text{agr} \text{ Agr4} \text{ vform Vf3}]) \rightarrow$
 $\quad v([\text{subcat_none} \text{ agr} \text{ Agr4} \text{ vform Vf3}])$
5. $vp([\text{agr} \text{ Agr4} \text{ vform Vf3}]) \rightarrow$
 $\quad v([\text{subcat_np} \text{ agr} \text{ Agr4} \text{ vform Vf3}]) \text{ np}()$

Grammar 4.20 A DCG version of Grammar 4.7

Grammar 4.20 A DCG version of Grammar 4.7

S features consist of "inv", "agr", "inv", and "vform", every rule using an S must specify all these features in the same order. In a realistic grammar this would be awkward, because there will be many possible features for each constituent, and each one must be listed in every occurrence. One way to solve this problem is to extend the

program that converts DCG rules to PROLOG rules so that it adds in any unspecified features with variable values. This way the user need only specify the features that are important.

>> [back](#)

o 4.8 Generalized Feature Systems and Unification Grammars

You have seen that feature structures are very useful for generalizing the notion of context-free grammars and transition networks. In fact, feature structures can be generalized to the extent that they make the context-free grammar unnecessary. The entire grammar can be specified as a set of constraints between feature structures. Such systems are often called unification grammars. This section provides an introduction to the basic issues underlying such formalisms.

The key concept of a unification grammar is the extension relation between two feature structures. A feature structure F1 extends, or is more specific than, a feature structure F2 if every feature value in F1 is specified in F2. For example, the feature structure

```
(CAT V  
ROOT cry)
```

extends the feature structure (CAT V), as its CAT value is V as required, and the ROOT feature is unconstrained in the latter feature structure. On the other hand, neither of the feature structures

```
(CAT V (CAT V
```

(CAT V
ROOT cry)

(CAT V
VFORM pres)

extend the other, because both lack information required by the other. In particular, the first lacks the VFORM feature required by the second, and the second lacks the ROOT feature required by the first.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 110]

Two feature structures **unify** if there is a feature structure that is an extension of both. The **most general unifier** is the minimal feature structure that is an extension of both. The most general unifier of the above two feature structures is

(CAT V
ROOT cry
VFORM pres)

Note that this is an extension of both original feature structures, and there is no smaller feature structure that is an extension of both. In contrast, the structures

(CAT V

AGR 3s)

(CAT V

AGR 3p)

do not unify. There can be no FS that is an extension of both because they specify contradictory AGR feature values.

This formalism can be extended to allow simple disjunctive values (for example, {3s 3p}) in the natural way. For example, (AGR 3s) extends (AGR {3s 3p}).

The notion of unification is all that is needed to specify a grammar, as all feature agreement checks and manipulations can be specified in terms of unification relationships. A rule such as $S \rightarrow^* NP VP$ in Grammar 4.7 could be expressed in a unification grammar as

$x_0 \rightarrow x_1 x_2$

$CAT_0 = S$

$CAT_1 = NP$

$CAT_2 = VP$

AGR₀ = **AGR**₁ = **AGR**₂

VFORM₀ = **VFORM**₂

This says that a constituent X0 can be constructed out of a sequence of constituents X1 and X2 if the CAT of X0 is 5, the CAT of X1 is NP, the CAT of X2 is VP, the AGR values of all three constituents are identical, and the VFORM values of constituents X0 and X2 are identical. If the CAT value is always specified, such rules can be abbreviated as follows:

S → **NP VP** **AGR** = **AGR**₁ = **AGR**₂

VFORM = **VFORM**₂

where the CAT values are used in the rule. Also, the 0 subscript is omitted. Using these abbreviations, a subpart of Grammar 4.7 is rewritten as the unification grammar in Grammar 4.21. Since such grammars retain the structure of context-free rules, the standard parsing algorithms can be used on unification grammars. The next section shows how to interpret the feature equations to build the constituents.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 111]

- | | |
|------------------------------|---|
| 1'. $S \rightarrow NP\ VP$ | $AGR = AGR_1 = AGR_2$
$VFORM = VFORM_2$ |
| 2'. $NP \rightarrow ART\ N$ | $AGR = AGR_1 = AGR_2$ |
| 8'. $VP \rightarrow V\ ADJP$ | $SUBCAT_1 = _adjp$
$VFORM = VFORM_1$
$AGR = AGR_1$ |
| 10'. $ADJP \rightarrow ADJ$ | |

Grammar 4.21 A unification grammar

Grammar 4.21 unification grammar

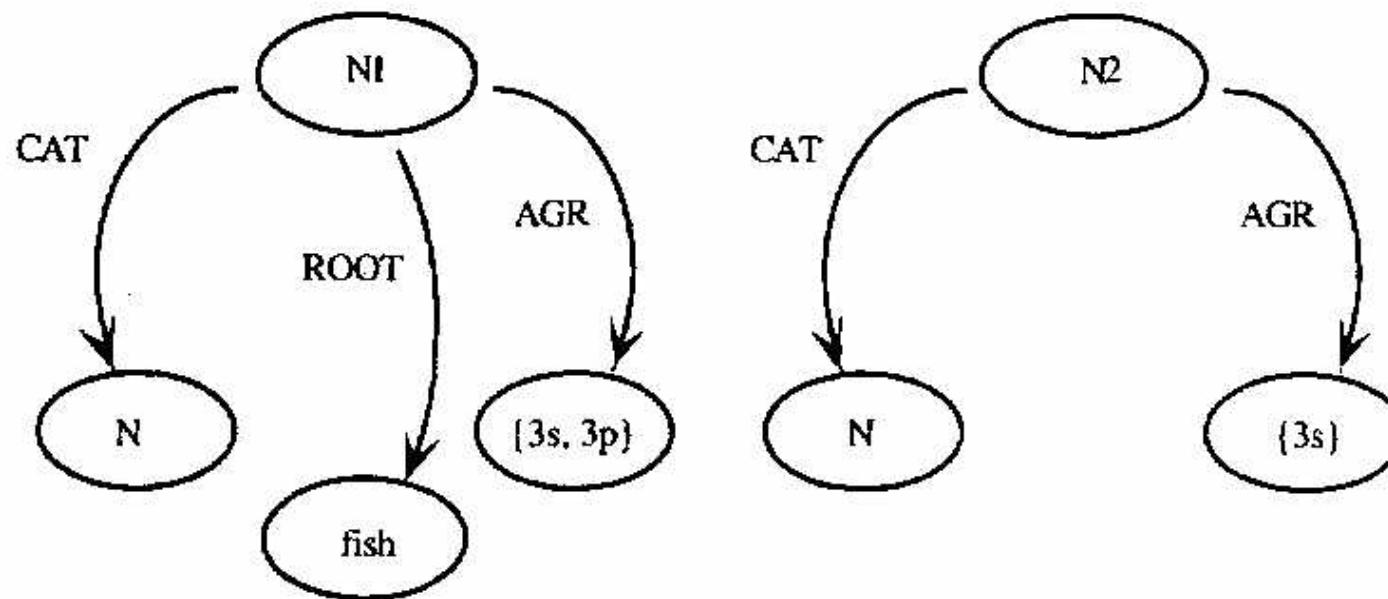


Figure 4.22 Two noun phrase DAGs

Figure 4.22 Two noun phrase DAGs

Formal Development: Feature Structures as DAGs

The unification-based formalism can be defined precisely by representing feature structures as directed acyclic graphs (DAGs). Each constituent and value is represented as a node, and the features are represented as labeled arcs. Representations of the following two constituents are shown in Figure 4.22:

N1: (CAT N

ROOT fish

AGR {3s 3p})

N2: (CAT N

AGR 3s)

The sources of a DAG are the nodes that have no incoming edges. Feature structure DAGs have a unique source node, called the root node. The DAG is said to be rooted by this node. The sinks of a DAG are nodes with no outgoing edges. The sinks of feature structures are labeled with an atomic feature or set of features (for example, {3s 3p}).

The unification of two feature structures is defined in terms of a graph-matching algorithm. This takes two rooted graphs and returns a new graph that is

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 112]

To unify a DAG rooted at node \mathbf{N}_i with a DAG rooted at node \mathbf{N}_j :

1. If \mathbf{N}_i equals \mathbf{N}_j , then return \mathbf{N}_i and succeed.
2. If both \mathbf{N}_i and \mathbf{N}_j are sink nodes, then if their labels have a non-null intersection, return a new node with the intersection as its label. Otherwise, the DAGs do not unify.
3. If \mathbf{N}_i and \mathbf{N}_j are not sinks, then create a new node \mathbf{N} . For each arc labeled F leaving \mathbf{N}_i to node \mathbf{NF}_i ,
 - 3a. If there is an arc labeled F leaving \mathbf{N}_i to node \mathbf{NF}_i , then recursively unify \mathbf{NF}_i and \mathbf{NF}_j . Build an arc labeled F from N to the result of the recursive call.
 - 3b. If there is no arc labeled F from \mathbf{N}_i , build an arc labeled F from N to \mathbf{NF}_i .
 - 3c. For each arc labeled F from \mathbf{N}_j to node NF, where there is no F arc leaving \mathbf{N}_j , create a new arc labeled F from N to \mathbf{NF}_j .

Figure 4.23 The graph unification algorithm

the unification of the two. The algorithm is shown in Figure 4.23. The result of applying this algorithm to nodes N1 and N2 in Figure 4.22 is the new constituent

N3: (CATN

ROOT fish

AGR 3s)

You should trace the algorithm on this example by hand to see how it works. This is a very simple case, as there is only one level of recursion. The initial call with the nodes N1 and N2 is handled in step 3, and each recursive call simply involves matching of sink nodes in step 2.

With this algorithm in hand, the algorithm for constructing a new constituent using the graph unification equations can be described. Once this is developed, you can build a parser using any standard parsing algorithm. The algorithm to build a new constituent of category C using a rule with feature equations of form $F_i = v$, where F_i indicates the **F** feature of the i'th subconstituent, is shown as Figure 4.24.

Consider an example. Assume the following two constituents shown in Figure 4.25 are defined already. In LISP notation, they are

ART1: (CAT

ROOT the

N1: (CATN

ROOT fish

ART1: (CAT

N1: (CATN

AGR {3s 3p})

AGRE {3s 3p})

The new NP will be built using rule 2 in Grammar 4.21. The equations are

$$\text{CAT}_0 = \text{NP}$$

$$\text{CAT}_1 = \text{ART}$$

$$\text{CAT}_2 = \text{N}$$

$$\text{AGR} = \text{AGR}_1 = \text{AGR}_2$$

The algorithm produces the constituent represented by the DAG in Figure 4.26.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 113]

Given a rule $X_0 \rightarrow X_1 \dots X_n$ and set of feature equations of form $F_i = V$, where SC_1, \dots, SC_n are the subconstituents corresponding to X_1, \dots, X_n , this algorithm builds a DAG that satisfies all the feature equations.

1. Create a node CC_0 to be the root of the new feature structure.
2. Make a copy of each DAG rooted by SC_i (call the new root of each CC_i), and add an arc labeled i from CC_0 to each CC_i .
3. For each feature equation of form $F_i = V$, where V is a value, follow the F link from node CC_i to a node N_i , and unify N_i with V .
4. For each feature equation (of form $F_i = G_j$),
 - 4a. If there is an F link from CC_i , and a G link from CC_j , then
 - i. follow the F link to node N_i and the G link to N_j ;
 - ii. unify N_i and N_j , using the graph unification algorithm, to create new node X ;
 - iii. change all arcs pointing to either N_i or N_j to point to X ;
 - 4b. If there is no F link from CC_i , but there is a G link from CC_j to node N_j , create an F link from CC_i to N_j ;
 - 4c. If there is no G link from CC_j , but there is an F link from CC_i to N_i , create a G link from CC_j to N_i .

Figure 4.24 An algorithm to build a new constituent using feature equations

Figure 4.24 An algorithm to build a new constituent using feature equations

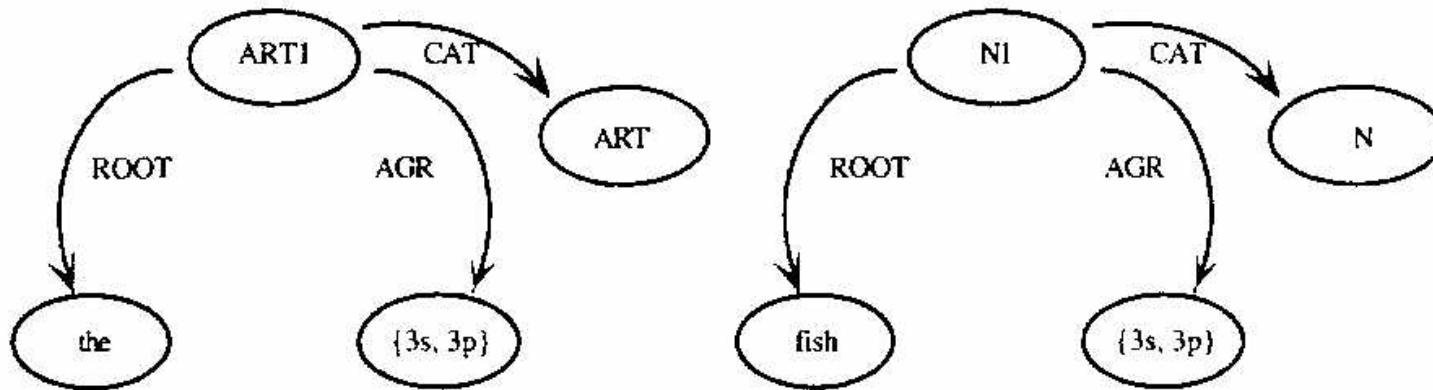


Figure 4.25 Lexical entries for *the* and *fish*

Figure 4.25 Lexical entries for the and fish

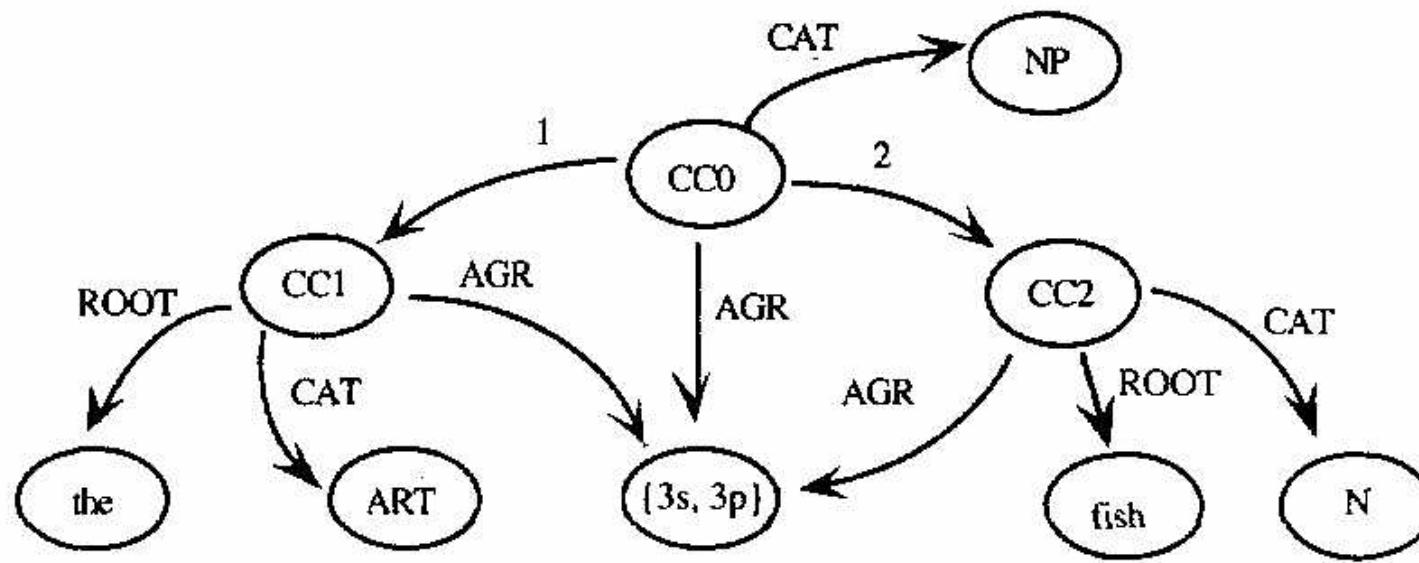


Figure 4.26 The graph for the NP *the fish*

Figure 4.26 The graph for the NP *the fish*

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 114]

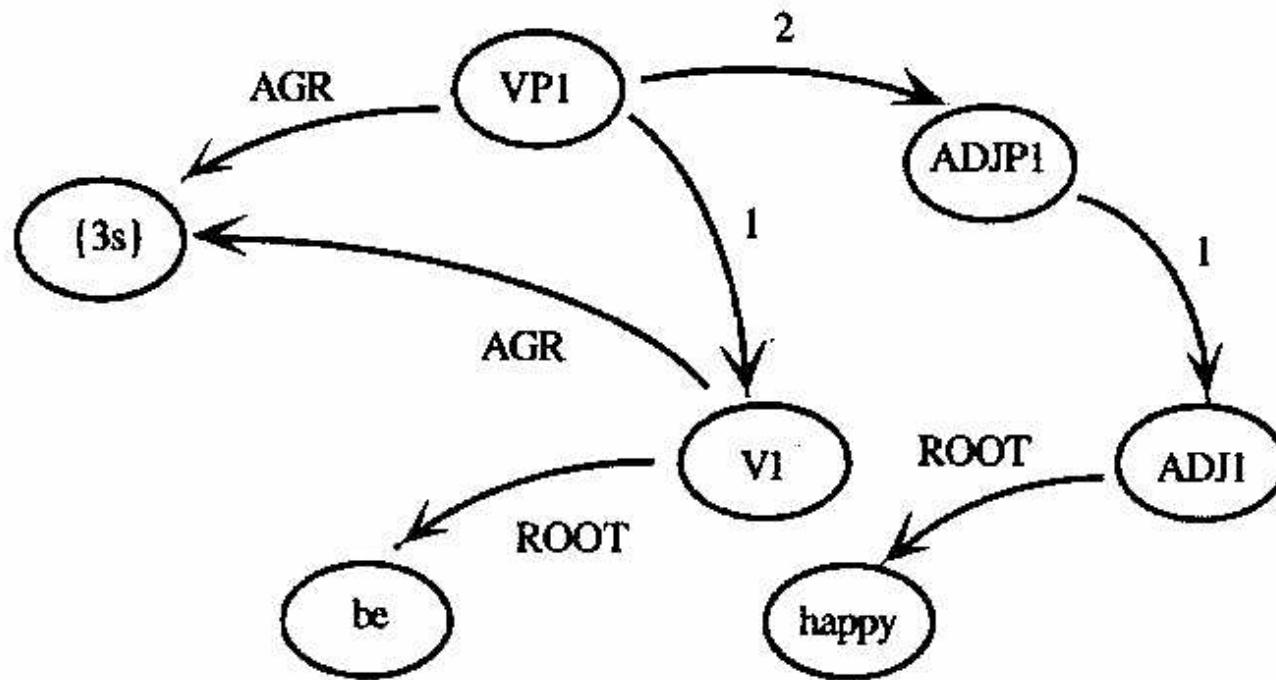


Figure 4.27 The analysis of the VP *is happy*

Figure 4.27 The analysis of the VP is happy

Continuing the example, assume that the VP is happy is analyzed similarly and is represented as in Figure 4.27. To simplify the graph, the CAT arcs are not shown. Figure 4.28 shows the analysis of The fish is happy constructed from rule 1 in Grammar 4.21. The value of the AGR slot is now the same node for S1, NF1, ART1, N1, VP1, and V1. Thus the value of the AGR feature of CCL, for instance, changed when the AGR features of NP1 and VP1 were unified.

So far, you have seen unification grammars used only to mimic the behavior of the augmented context-free grammars developed earlier. But the unification framework is considerably richer because there is no requirement that rules be based on syntactic categories. For example, there is a class of phrases in English called **predicative** phrases, which can occur in sentences of the form

NP be _

This includes prepositional phrases (He is in the house), noun phrases (He is a traitor), and adjective phrases (He is happy). Grammars often include a binary feature, say **PRED**, that is true of phrases that can be used in this way. In a standard CFG you would need to specify a different rule for each category, as in

VP \rightarrow (V **ROOT** be) (NP **PRED** +)

VP \rightarrow (V **ROOT** be) (PP **PRED** +)

VP \rightarrow (V **ROOT** be) (ADJP **PRED** +)

With the unification grammar framework, one rule handles all the categories, namely

X0 \rightarrow X1 X2

CAT₀ = VP

CAT₁ = V

ROOT₁ = be

PRED₂ = +

in which any constituent X2 with the +PRED feature is allowed.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 115]

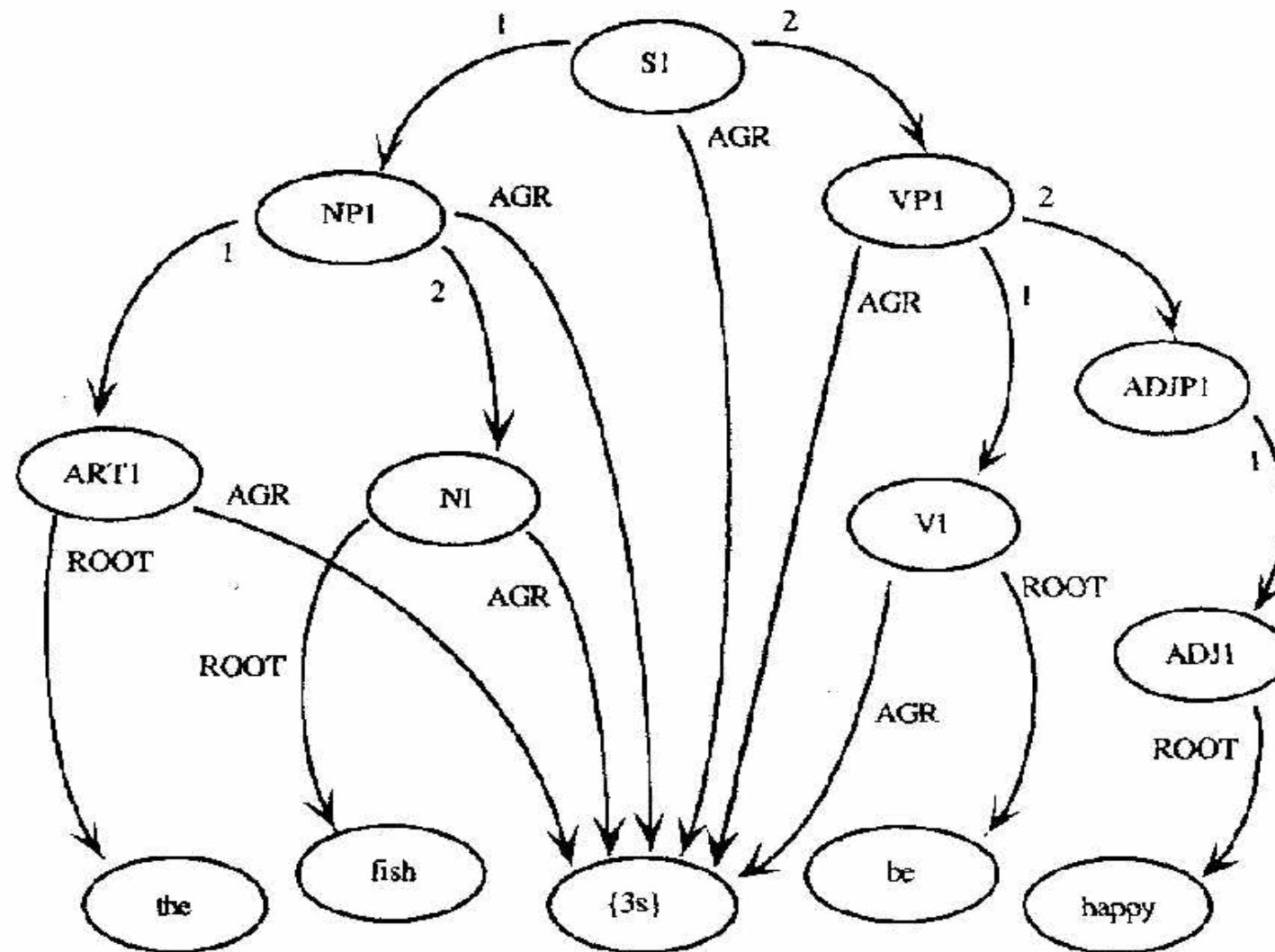


Figure 4.28 An analysis of *The fish is happy.*

Figure 4.28 An analysis of The fish is happy.

Of course, if the categories of constituents are not specified, it is not so clear how to adapt the standard CFG parsing algorithms to unification grammar. It can be shown that as long as there is a finite subset of features such that at least one of the features is specified in every constituent, the unification grammar can be converted into a context-free grammar. This set of features is sometimes called the **context-free backbone** of the grammar.

Another powerful feature of unification grammars is that it allows much more information to be encoded in the lexicon. In fact, almost the entire grammar can be encoded in the lexicon, leaving very few rules in the grammar. Consider all the rules that were introduced earlier to deal with different verb subcategorizations. These could be condensed to a few rules: one for verbs that sub-categorize for one subconstituent, another for verbs that subcategorize for two, and so on. The actual category restrictions on the complement structure would be encoded in the lexicon entry for the verb. For example, the verb put might have a lexicon entry as follows:

```
put: (CAT V SUBCAT (FIRST (CAT NP) SECOND (CAT PP LOC +)))
```

The general rule for verbs that subcategorize for two constituents would be

VP → V X₂ X₃

2 = FIRST_{SUBCAT1}

3 = SECOND_{SUBCAT1}

This rule would accept any sequence V X₂ X₃, in which X₂ unifies with the FIRST feature of the SUBCAT feature of X₁ and X₃ unifies with the SECOND feature of the SUBCAT feature of X₁. If V is the verb put, this rule would require X₂ to unify with (CAT NP) and X₃ to unify with (CAT PP LOC +), as desired. Of course, this same rule would put completely different constraints on X₂ and X₃ for a verb like want, which would require X₃ to unify with (CAT VP VFORM inf).

Such techniques can dramatically reduce the size of the grammar. For instance, a reasonable grammar would require at least 40 rules to handle verb subcategorizations. But no rule involves more than three subconstituents in the verb complement. Thus these 40 rules could be reduced to four in the unification grammar, including a rule for null complements. In addition, these same rules could be reused to handle all nouns and adjective subcategorizations, if you generalize the category restrictions on X₁.

>> [back](#)

Summary

This chapter has extended the grammatical formalisms introduced in Chapter 3 by adding features to each constituent and augmenting the grammatical rules. Features are essential to enable the construction of wide-coverage grammars of natural language. A useful set of features for English was defined, and morphological analysis techniques for mapping words to feature structures (that is, constituents) were developed. Several different forms of augmentation were examined. The first allowed context-free rules to specify feature agreement restrictions in addition to the basic category, producing augmented context-free grammars. The standard parsing algorithms described in Chapter 3 can be extended to handle such grammars. The second technique produced augmented transition networks, in which different procedural tests and actions could be defined on each arc to manipulate feature structures. The final method was based on the unification of feature structures and was used to develop unification grammars.

>> [back](#)

Related Work and Further Readings

Augmented CFGs have been used in computational models since the introduction of attribute grammars by Knuth (1968), who employed them for parsing programming languages. Since then many systems have utilized annotated rules of some form. Many early systems relied on arbitrary LISP code for annotations, although the types of operations commonly implemented were simple feature

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 117]

BOX 4.3 Lexical Functional Grammar

BOX 4.3 Lexical Functional Grammar

A linguistic theory that has been influential in the development of computational formalisms is lexical functional grammar (Kaplan and Bresnan, 1982), usually abbreviated as LFG. LFG can be viewed as a type of unification grammar. A typical LFG rule is as follows:

$$S \rightarrow NP \quad VP \\ (\uparrow \text{SUBJ}) = \downarrow \quad \uparrow = \downarrow$$

The up arrow (\uparrow) indicates the constituent named on the left-hand side of the rule (the *S* constituent); the down arrow (\downarrow) indicates the constituent to which the annotation is attached. Thus this rule is equivalent to the following in the notation:

$$S \rightarrow NP \quad VP \\ \text{SUBJ} = 1 \\ 0 = 2$$

Note the unification of the entire *S* and *VP* structure, making them the same constituent. Computationally, this can be viewed as an efficient way to transfer all the registers from the *VP* to the *S*, but it has linguistic implications as well, as discussed in Kaplan and Bresnan (1982). The effect is that any further modification to the *S* or *VP* structure will affect both, since they are now the same constituent.

LFGs encode most of their information in the lexicon. In particular, lexical entries may indicate which slots they will fill in the constituent that contains them. For example, the entries for *a*, *the*, and *bird* might be as follows:

<i>a</i>	ART	(\uparrow SPEC) = INDEF (\uparrow AGR) = (3s)
<i>the</i>	ART	(\uparrow SPEC) = DEF
<i>bird</i>	N	(\uparrow AGR) = (3s) (\uparrow HEAD) = BIRD

The up-arrow annotations actually fill in slots in the *NP* structure. Using the rule

$$NP \rightarrow ART \ N$$

on the phrase *a bird* would result in the *AGR* feature of the *NP* being set to 3s when the word *a* is parsed, and then this value is unified with 3s to check number agreement when the noun *bird* is parsed. With the article *the*, no number agreement is checked, since the word *the* does not set the *AGR* feature of its *NP*.

tests and structure building along the lines of those discussed here. Examples of such systems are Sager (1981) and Robinson (1982).

The particular features used in this chapter and the techniques for augmenting rules are loosely based on work in the generalized phrase structure grammar (GPSG) tradition (Gazdar, Klein, Pullum, and Sag, 1985). GPSG introduces a finite set of feature values that can be attached to any grammatical symbol. These play much the same role as the annotations developed in this book except that there are no feature names. Rather since all values are unique, they define both

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 118]

the feature type and the value simultaneously. For example, a plural noun phrase in the third person with gender female would be of the grammatical type

NP [PL, 3, F]

Since the number of feature values is finite, the grammar is formally equivalent to a context-free grammar with a symbol for every combination of categories and features. One of the important contributions of GPSG, however, is the rich structure it imposes on the propagation of features. Rather than using explicit feature equation rules, GPSG relies on general principles of feature propagation that apply to all rules in the grammar. A good example of such a general principle is the head feature convention, which states that all the head features on the parent constituent must be identical to its head constituent. Another general principle enforces agreement restrictions between constituents. Some of the conventions introduced in this chapter to reduce the number of feature equations that must be defined by hand for each rule are motivated by these theoretical claims. An excellent survey of GPSG and LFG is found in Sells (1985).

The ATN framework described here is drawn from the work described in Woods (1970; 1973) and Kaplan (1973). A good survey of ATNs is Bates (1978). Logic programming approaches to natural language parsing originated in the early 1970s with work by Colmerauer (1978). This approach is perhaps best described in a paper by Pereira and Warren (1980) and in Pereira and Shieber (1987). Other interesting developments can be found in McCord (1980) and Pereira (1981). A good recent example is Alshawi (1992).

The discussion of unification grammars is based loosely on work by Kay (1982) and the PATR-II system (Shieber, 1984; 1986). There is a considerable amount of active research in this area. An excellent survey of the area is found in Shieber (1986). There is also a growing body of work on formalizing different forms of feature structures. Good examples are Rounds (1988), Shieber (1992), and Johnson (1991).

>> [back](#)

Exercises for Chapter 4

1. (easy) Using Grammar 4.7, draw the complete charts resulting from parsing the two sentences The man cries and He wants to be happy, whose final analyses are shown in Figure 4.9. You may use any parsing algorithm you want, but make sure that you state which one you are using and that the final chart contains every completed constituent built during the parse.
2. (medium) Define the minimal set of lexicon entries for the following verbs so that, using the morphological analysis algorithm, all standard forms of the verb are recognized and no illegal forms are inadvertently produced. Discuss any problems that arise and assumptions you make, and suggest modifications to the algorithms presented here if needed.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 119]

Base	Present Forms	Past	Past-Participle	Present-Participle
go	go, goes	went	gone	going
sing	sing, sings	sang	sung	singing
bid	bid, bids	bid	bidden	bidding

3. (medium) Extend the lexicon in Figure 4.6 and Grammar 4.7 so that the following two sentences are accepted:

He was sad to see the dog cry.

He saw the man saw the wood with the saw.

Justify your new rules by showing that they correctly handle a range of similar cases. Either implement and test your extended grammar using the supplied parser, or draw out the full chart that would be constructed for each sentence by a top-down chart parser.

4. (medium)

- a. Write a grammar with features that will successfully allow the following phrases as noun phrases:

three o'clock

quarter after eight

ten minutes to six

seven thirty-five

half past four

but will not permit the following:

half to eight three

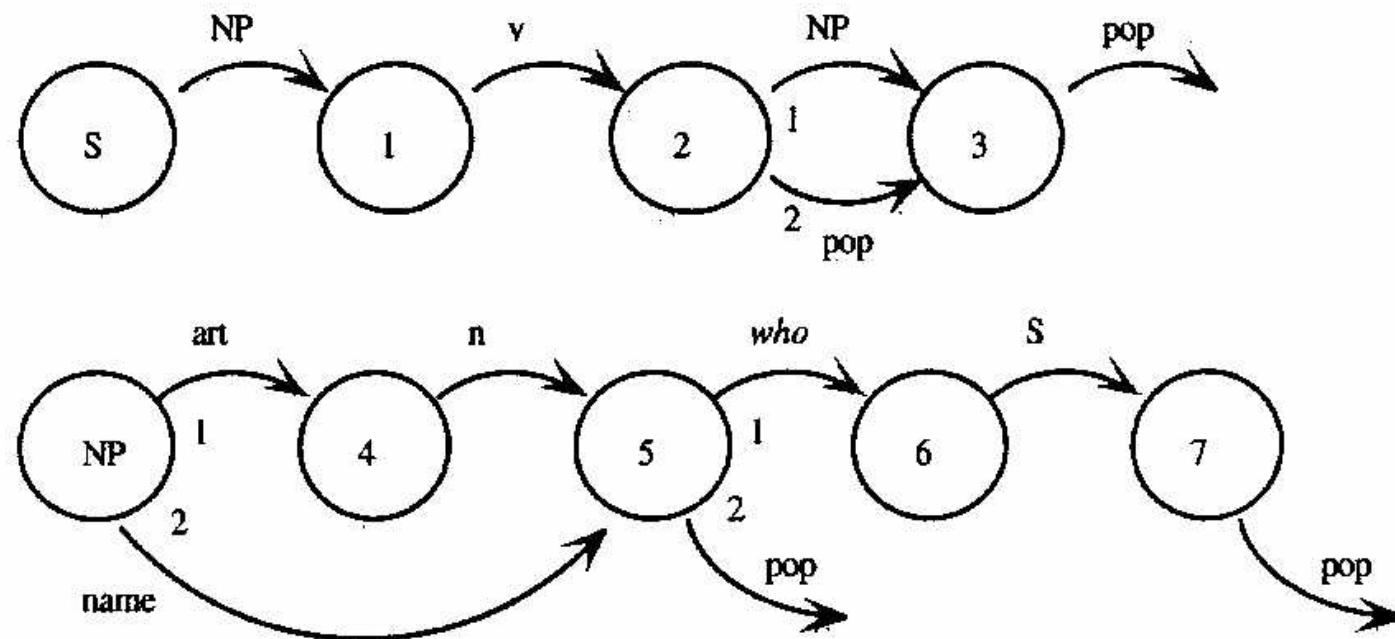
twenty o'clock

ten forty-five after six

Specify the feature manipulations necessary so that once the parse is completed, two features, HOUR and MINUTES, are set in the NP constituent. If this requires an extension to the feature mechanism, carefully describe the extension you assume.

- b. Choose two other forms of grammatical phrases accepted by the grammar. Find an acceptable phrase not accepted by your grammar. If any nongrammatical phrases are allowed, give one example.
5. (medium) English pronouns distinguish case. Thus I can be used as a subject, and me can be used as an object. Similarly, there is a difference between he and him, we and us, and they and them. The distinction is not made for the pronoun you. Specify an augmented context-free grammar and lexicon for simple subject-verb-object sentences that allows only appropriate pronouns in the subject and object positions and does number agreement between the subject and verb. Thus it should accept (hit him, but not me love you. Your grammar should account for all the pronouns mentioned in this question, but it need have only one verb entry and need cover no other noun phrases but pronouns.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 120]



- Specify some words by category and give four structurally different sentences accepted by this network.
- Specify an augmentation for this network in the notation defined in this chapter so that sentences with the main verb give are allowed only if the subject is animate, and sentences with the main verb be may take either an animate or inanimate subject. Show a lexicon containing a few words that can be used to demonstrate the network's selectivity.

7. (medium) Using the following unification grammar, draw the DAGs for the two NP structures as they are when they are first constructed by the parser, and then give the DAG for the complete sentence (which will include all subconstituents of S as well) The fish is a large one. You may assume the lexicon in Figure 4.6, but define lexical entries for any words not covered there.

1. $S \rightarrow NP\ VP$

INV = -

VFORM₂ = pres

AGR = **AGR**₁ = **AGR**₂

AGR = **AGR**₁ = **AGR**₂

AGR = **AGR**₁ = **AGR**₃

VFORM = **VFORM**₁

AGR = **AGR**₁ = **AGR**₂

ROOT = BE1

2. $NP \rightarrow ART\ N$

3. $NP \rightarrow ART\ ADJ\ N$

4. $VP \rightarrow V\ NP$

>> [back](#)

Allen 1995: Natural Language Understanding

	<u>Contents</u>	<u>Preface</u>	<u>Introduction</u>	
<u>previous chapter</u>	<u>Part I Syntactic Processing</u>	<u>Part II Semantic Interpretation</u>	<u>Part III - Context / World Knowledge</u>	<u>next chapter</u>
	<u>Appendices</u>	<u>Bibliography</u>		
	<u>Summaries</u>	<u>Further Readings</u>	<u>Exercises</u>	

Chapter 5: Grammars for Natural Language

[5.1 Auxiliary Verbs and Verb Phrases](#)

[5.2 Movement Phenomena in Language](#)

[5.3 Handling Questions in Context-Free Grammars](#)

[o 5.4 Relative Clauses](#)

[5.5 The Hold Mechanism in ATNs](#)

[5.6 Gap Threading](#)

[Summary](#)

[Related Work and Further Readings](#)

[Exercises for Chapter 5](#)

[Allen 1995: Chapter 5 - Grammars for Natural Language 123]

Augmented context-free grammars provide a powerful formalism for capturing many generalities in natural language. This chapter considers several aspects of the structure of English and examines how feature systems can be used to handle them. Section 5.1 discusses auxiliary verbs and introduces features that capture the ordering and agreement constraints. Section 5.2 then discusses the general class of problems often characterized as movement phenomena. The rest of the chapter examines various approaches to handling movement. Section 5.3 discusses handling yes/no questions and wh-questions, and Section 5.4 discusses relative clauses. The remaining sections discuss alternative approaches. Section 5.5 discusses the use of the hold mechanism in ATNs and Section 5.6 discusses gap threading in logic grammars.

>> [back](#)

5.1 Auxiliary Verbs and Verb Phrases

English sentences typically contain a sequence of auxiliary verbs followed by a main verb, as in the following:

I can see the house.

I will have seen the house.

I was watching the movie.

I should have been watching the movie.

These may at first appear to be arbitrary sequences of verbs, including *have*, *be*, *do*, *can*, *will*, and so on, but in fact there is a rich structure. Consider how the auxiliaries constrain the verb that follows them. In particular, the auxiliary *have* must be followed by a past participle form (either another auxiliary or the main verb), and the auxiliary *be* must be followed by a present participle form, or, in the case of passive sentences, by the past participle form. The auxiliary *do* usually occurs alone but can accept a base form following it (*I did eat my carrots!*). Auxiliaries such as *can* and *must* must always be followed by a base form. In addition, the first auxiliary

(or verb) in the sequence must agree with the subject in simple declarative sentences and be in a finite form (past or present). For example, **I going*, **we be gone*, and **they am* are all unacceptable.

This section explores how to capture the structure of auxiliary forms using a combination of new rules and feature restrictions. The principal idea is that auxiliary verbs have subcategorization features that restrict their verb phrase complements. To develop this, a clear distinction is made between auxiliary and main verbs. While some auxiliary verbs have many of the properties of regular verbs, it is important to distinguish them. For example, auxiliary verbs can be placed before an adverbial *not* in a sentence, whereas a main verb cannot:

I am not going!

You did not try it.

He could not have seen the car.

[Allen 1995: Chapter 5 - Grammars for Natural Language 124]

Auxiliary	COMPFORM	Construction	Example
modal	base	modal	<i>can see the house</i>
have	pastprt	perfect	<i>have seen the ^{house} house</i>
be	ing	progressive	<i>is lifting the box</i>
be	pastprt	passive	<i>was seen by the crowd</i>

Figure 5.1 The COMPFORM restrictions for auxiliary verbs

In addition, only auxiliary verbs can precede the subject NP in yes/no questions:

Did you see the car? Can I try it?

*** Eat John the pizza?**

In contrast, main verbs may appear as the sole verb in a sentence, and if made into a yes/no question require the addition of the auxiliary *do*:

I ate the pizza.

Did I eat the pizza?

The boy climbed in the window.

Did the boy climb in the window?

I have a pen.

Do I have a pen?

The primary auxiliaries are based on the root forms "*be*" and "*have*". The other auxiliaries are called modal auxiliaries and generally appear only in the finite tense forms (simple present and past). These include the following verbs organized in pairs corresponding roughly to present and past verb forms "*do (did)*", "*can (could)*", "*may (might)*", "*shall (should)*", "*will (would)*", "*must*", "*need*", and "*dare*". In addition, there are phrases that also serve a modal auxiliary function, such as "*ought to*", "*used to*", and "*be going to*".

Notice that "*have*" and "*be*" can be either an auxiliary or a main verb by these tests. Because they behave quite differently, these words have different lexical entries as auxiliaries and main verbs to allow for different properties; for example, the auxiliary "*have*" requires a past-participle verb phrase to follow it, whereas the verb "*have*" requires an NP complement.

As mentioned earlier, the basic idea for handling auxiliaries is to treat them as verbs that take a VP as a complement. This VP may itself consist of another auxiliary verb and another VP, or be a VP headed by a main verb. Thus, extending Grammar 4.7 with the following rule covers much of the behavior:

```
VP -> (AUX COMPFORM ?s) (VP VFORM ?s)
```

The COMPFORM feature indicates the VFORM of the VP complement. The values of this feature for the auxiliaries are shown in Figure 5.1.

There are other restrictions on the auxiliary sequence. In particular, auxiliaries can appear only in the following order:

Modal + have + be (progressive) + be (passive)

The song might have been being played as
they left

[Allen 1995: Chapter 5 - Grammars for Natural Language 125]

To capture the ordering constraints, it might seem that you need eight special rules: one for each of the four auxiliary positions plus four that make each optional. But it turns out that some restrictions can fall out from the feature restrictions. For instance, since modal auxiliaries do not have participle forms, a modal auxiliary can never follow "have" or "be" in the auxiliary sequence. For example, the sentence

* **He has might see the movie already.**

violates the subcategorization restriction on "have". You might also think that the auxiliary "have" can never appear in its participle forms, as it must either be first in the sequence (and be finite) or follow a modal and be an infinitive. But this is only true for simple matrix clauses. If you consider auxiliary sequences appearing in VP complements for certain verbs, such as "regret", the participle forms of "have" as an auxiliary can be required, as in

I regret having been chosen to go.

As a result, the formulation based on subcategorization features over-generates. While it accepts any legal sequence of auxiliaries, it would also accept

* I must be having been singing.

This problem can be solved by adding new features that further restrict the complements of the auxiliary "be" so that they do not allow additional auxiliaries (except "be" again for the passive). A binary head feature MAIN could be introduced that is + for any main verb, and - for auxiliary verbs. This way we can restrict the VP complement for "be" as follows:

```
VP -> AUX[be] VP[ing, +main]
```

The lexical entry for "be" would then have to be changed so that the original auxiliary rule does not apply. This could be done by setting its CQMPFORM feature to -. The only remaining problem is how to allow the passive. There are several possible approaches. We could, for instance, treat the "be" in the passive construction as a main verb form rather than an auxiliary. Another way would be simply to add another rule allowing a complement in the passive form, using a new binary feature PASS, which is + only if the VP involves passive:

```
VP -> AUX[be] VP[ing, +pass]
```

The passive rule would then be:

```
VP[+pass] -> AUX [be] VP[pastprt, main]
```

While these new rules capture the ordering constraints well, for the sake of keeping examples short, we will generally use the first rule presented for handling auxiliaries throughout. While it overgenerates, it will not cause any problems for the points that are illustrated by the examples.

[Allen 1995: Chapter 5 - Grammars for Natural Language 126]

can:	(CAT AUX MODAL + VFORM pres AGR {1s 2s 3s 1p 2p 3p} COMPFORM base)	could:	(CAT AUX MODAL + VFORM {pres past} AGR {1s 2s 3s 1p 2p 3p} COMPFORM base)
do:	(CAT AUX MODAL + VFORM pres AGR {1s 2s 1p 2p 3p} COMPFORM base)	did:	(CAT AUX MODAL + VFORM past AGR {1s 2s 3s 1p 2p 3p} COMPFORM base)
be:	(CAT AUX VFORM base ROOT be COMPFORM ing)	have:	(CAT AUX VFORM base ROOT have COMPFORM pastprt)

Figure 5.2 Lexicon entries for some auxiliary verbs

Figure 5.2 Lexicon entries for some auxiliary verbs

A lexicon for some of the auxiliary verbs is shown in Figure 5.2. All the irregular forms would need to be defined as well.

o Passives

The rule for passives in the auxiliary sequence discussed in the previous section solves only one of the problems related to the passive construct. Most verbs that include an NP in their complement allow the passive form. This form involves using the normal "object position" NP as the first NP in the sentence and either omitting the NP usually in the subject position or putting it in a PP with the preposition "by". For example, the active voice sentences

I will hide my hat in the drawer.

I hid my hat in the drawer.

I was hiding my hat in the drawer.

can be rephrased as the following passive voice sentences:

My hat will be hidden in the drawer.

My hat was hidden in the drawer.

My hat was being hidden in the drawer.

The complication here is that the VP in the passive construction is missing the object NP. One way to solve this problem would be to add a new grammatical rule for every verb subcategorization that is usable only for passive forms, namely all rules that allow an NP to follow the verb. A program can easily be written that would automatically generate such passive rules given a grammar. A

[Allen 1995: Chapter 5 - Grammars for Natural Language 127]

1. $S[-\text{inv}] \rightarrow (\text{NP } AGR ?a) (\text{VP } [fin] AGR ?a)$
2. $\text{VP} \rightarrow (\text{AUX } COMPFORM ?v) (\text{VP } VFORM ?v)$
3. $\text{VP} \rightarrow \text{AUX}[be] \text{ VP}\{\text{ing}, +\text{main}\}$
4. $\text{VP} \rightarrow \text{AUX}[be] \text{ VP}\{\text{ing}, +\text{pass}\}$
5. $\text{VP}\{+\text{pass}\} \rightarrow \text{AUX}[be] \text{ VP}\{\text{pastprt}, \text{main}, +\text{passgap}\}$
6. $\text{VP}\{-\text{passgap}, +\text{main}\} \rightarrow V[_none]$
7. $\text{VP}\{-\text{passgap}, +\text{main}\} \rightarrow V[_np] \text{ NP}$
8. $\text{VP}\{+\text{passgap}, +\text{main}\} \rightarrow V[_np]$
9. $\text{NP} \rightarrow (\text{ART } AGR ?a) (NAGR ?a)$
10. $\text{NP} \rightarrow NAME$
11. $\text{NP} \rightarrow PRO$

Head features for S, VP: AGR and VFORM

Head features for NP: AGR

Figure 5.3 A fragment handling auxiliaries including passives

Figure 5.3 A fragment handling auxiliaries including passives

new binary head feature, PASSGAP, is defined that is + only if the constituent is missing the object NP. As usual, this feature would default to - if it is not specified in the left-hand side of a rule. The rule for a simple _np subcategorization in Grammar 4.5 would then be realized as two rules in the new grammar:

`VP[-passgap] → V[_np] NP`

`VP[+passgap] → V[_np]`

Since the PASSGAP feature defaults to -, the only way the PASSGAP feature can become + is by the use of passive rules. Similarly, VP rules with lexical heads but with no NP in their complement are -PASSGAP, because they cannot participate in the passive construction. Figure 5.3 shows a fragment of Grammar 4.5 extended to handle auxiliaries and passives. The rules are as developed above except for additional variables in each rule to pass features around appropriately.

Figure 5.4 shows the analysis of the active voice sentence "*Jack can see the dog*" and the passive sentence "*Jack was seen*". Each analysis uses rule 1 for the S and rule 7 for the VP. The only difference is that the active sentence must use auxiliary rule 2 and NP rule 9, while the passive sentence must use auxiliary rule 5 and NP rule 8.

>> [back](#)

5.2 Movement Phenomena in Language

Many sentence structures appear to be simple variants of other sentence structures. In some cases, simple words or phrases appear to be locally reordered; sentences are identical except that a phrase apparently is moved from its expected

[Allen 1995: Chapter 5 - Grammars for Natural Language 128]

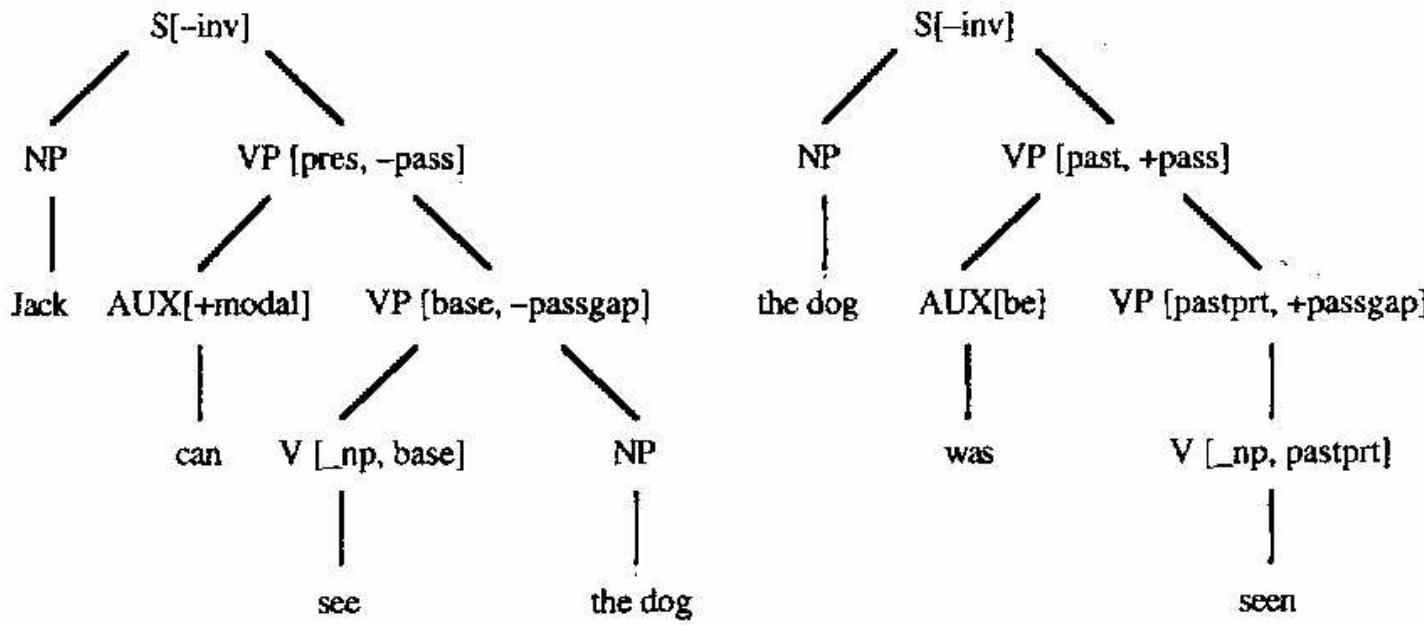


Figure 5.4 An active and a passive form sentence

Figure 5.4 An active and a passive form sentence

position in a basic sentence. This section explores techniques for exploiting these generalities to cover questions in English.

As a starting example, consider the structure of yes/no questions and how they relate to their assertional counterpart. In particular, consider the following examples:

Jack is giving Sue a back He will run in the marathon
rub. next year.

Is Jack giving Sue a back Will he run in the marathon
rub? next year?

As you can readily see, yes/no questions appear identical in structure to their assertional counterparts except that the subject NPs and first auxiliaries have swapped positions. If there is no auxiliary in the assertional sentence, an auxiliary of root "do", in the appropriate tense, is used:

John went to the store. Henry goes to school every day.

Did John go to the store? Does Henry go to school every
day?

Taking a term from linguistics, this rearranging of the subject and the auxiliary is called subject-aux inversion.

Informally, you can think of deriving yes/no questions from assertions by moving the constituents in the manner just described. This is an example of local (or bounded) movement. The movement is considered local because the rearranging of the constituents is specified precisely within the scope of a limited number of rules. This contrasts

with unbounded movement, which occurs in wh-questions. In cases of unbounded movement, constituents may be moved arbitrarily far from their original position.

For example, consider the wh-questions that are related to the assertion

The fat man will angrily put the book in the corner.

If you are interested in who did the action, you might ask one of these questions:

[Allen 1995: Chapter 5 - Grammars for Natural Language 129]

Which man will angrily put the book in the corner?

Who will angrily put the book in the corner?

On the other hand, if you are interested in how it is done, you might ask one of the following questions:

How will the fat man put the book in the corner?

In what way will the fat man put the book in the corner?

If you are interested in other aspects, you might ask one of these questions:

What will the fat man angrily put in the corner?

Where will the fat man angrily put the book?

In what corner will the fat man angrily put the book?

What will the fat man angrily put the book in?

Each question has the same form as the original assertion, except that the part being questioned is removed and replaced by a wh-phrase at the beginning of the sentence. In addition, except when the part being queried is the subject NP, the subject and the auxiliary are apparently inverted, as in yes/no questions. This similarity with yes/no questions even holds for sentences without auxiliaries. In both cases, a "do" auxiliary is inserted:

I found a bookcase.

Did I find a bookcase?

What did I find?

Thus you may be able to reuse much of a grammar for yes/no questions for wh-questions. A serious problem remains, however, concerning how to handle the fact that a constituent is missing from someplace later in the sentence. For example, consider the italicized VP in the sentence

What will the fat man *angrily put in the corner?*

While this is an acceptable sentence, "*angrily put in the corner*" does not appear to be an acceptable VP because you cannot allow sentences such as **I angrily put in the corner*". Only in situations like wh-questions can such a VP be allowed, and then it is allowed only if the wh-constituent is of the right form to make a legal VP if it were inserted in the sentence. For example, "*What will the fat man angrily put in the corner?*" is acceptable, but **Where will the fat man angrily put in the corner?*" is not.

If you constructed a special grammar for VPs in wh-questions, you would need a separate grammar for each form of VP and each form of missing constituent. This would create a significant expansion in the size of the grammar.

This chapter describes some techniques for handling such phenomena concisely. In general, they all use the same type of approach. The place where a subconstituent is missing is called the **gap**, and the constituent that is moved is called the **filler**. The techniques that follow all involve ways of allowing gaps in constituents when there is an appropriate filler available. Thus the analysis of the

[Allen 1995: Chapter 5 - Grammars for Natural Language 130]

VP in a sentence such as "*What will the fat man angrily put in the corner?*" is parsed as though it were "*angrily put what in the corner*", and the VP in the sentence "*What will the fat man angrily put the book in?*" is parsed as though the VP were "*angrily put the book in what*".

There is further evidence for the correctness of this analysis. In particular, all the well-formedness tests, like subject-verb agreement, the case of pronouns (who vs. whom), and verb transitivity, operate as though the wh-term were actually filling the gap. For example, you already saw how it handled verb transitivity. The question "*What did you put in the cupboard?*" is acceptable even though "put" is a transitive verb and thus requires an object. The object is a gap filled by the wh-term, satisfying the transitivity constraint. Furthermore, a sentence where the object is explicitly filled is unacceptable:

* **What did you put the bottle in the cupboard?**

This sentence is unacceptable, just as any sentence with two objects for the verb "put" would be unacceptable. In effect it is equivalent to

* **You put what the bottle in the cupboard?**

Thus the standard transitivity tests will work only if you assume the initial wh-term can be used to satisfy the constraint on the standard object NP position.

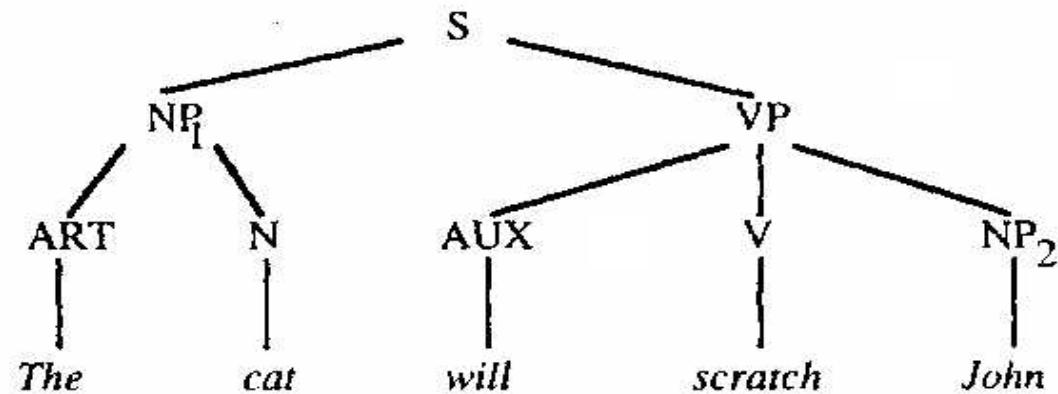
Many linguistic theories have been developed that are based on the intuition that a constituent can be moved from one location to another. As you explore these techniques further, you will see that significant generalizations can be made that greatly simplify the construction of a grammar. **Transformational grammar** (see Box 5.1) was based on this model. A context-free grammar generated a base sentence; then a set of transformations converted the resulting syntactic tree into a different tree by moving constituents. Augmented transition networks offered a new formalism that captured much of the behavior in a more computationally effective manner. A new structure called the **hold list** was introduced that allowed a constituent to be saved and used later in the parse by a new arc called the virtual (VIR) arc. This was the predominant computational mechanism for quite some time. In the early 1980s, however, new techniques were developed in linguistics that strongly influenced current computational systems.

The first technique was the introduction of **slash** categories, which are complex nonterminals of the form X/Y and stand for a constituent of type X with a subconstituent of type Y missing. Given a context-free grammar, there is a simple algorithm that can derive new rules for such complex constituents. With this in hand, grammar writers have a convenient notation for expressing the constraints arising from unbounded movement. Unfortunately, the size of the resulting grammar can be significantly larger than the original grammar. A better approach uses the feature system. Constituents are stored in a special feature called GAP and are passed from constituent to subconstituent to allow movement. In this analysis the constituent S/NP is shorthand for an S constituent with the GAP feature NP. The resulting system does not require expanding the size of

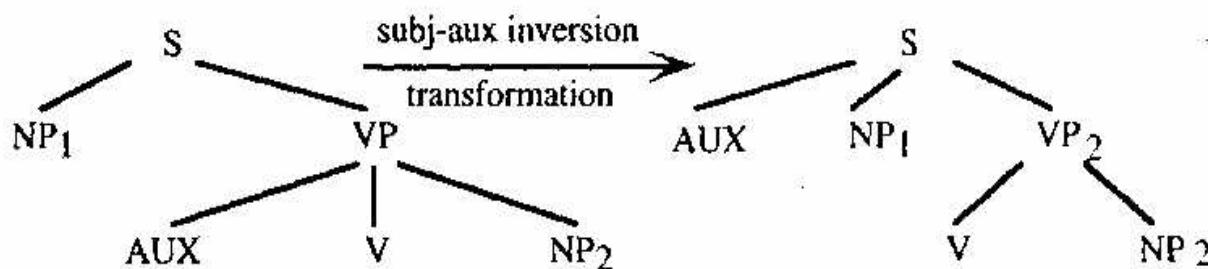
[Allen 1995: Chapter 5 - Grammars for Natural Language 131]

BOX 5.1 Movement in Linguistics

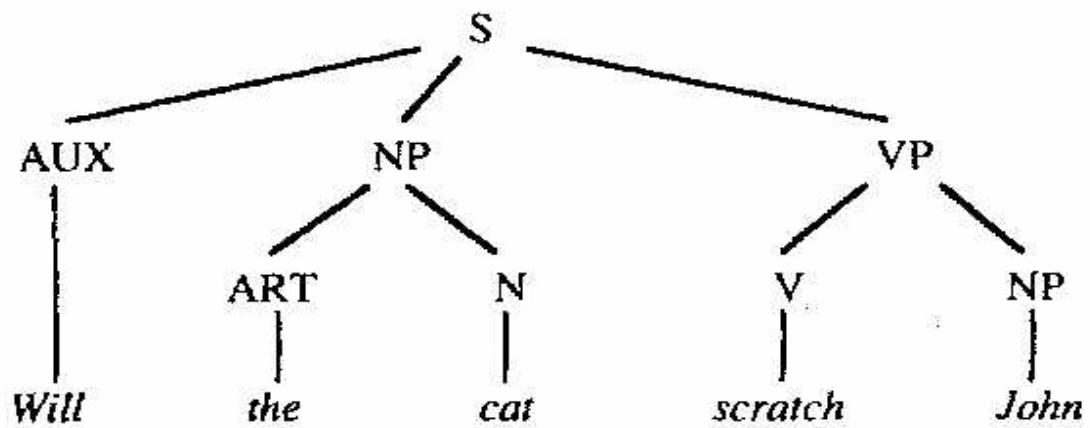
The term *movement* arose in transformational grammar (TG). TG posited two distinct levels of structural representation: surface structure, which corresponds to the actual sentence structure, and deep structure. A CFG generates the deep structure, and a set of transformations map the deep structure to the surface structure. For example, the deep structure of "*Will the cat scratch John?*" would be:



The yes/no question is then generated from this deep structure by a transformation expressed schematically as follows



With this transformation the surface form will be



In early TG (Chomsky, 1965), transformations could do many operations on trees, including moving, adding, and deleting constituents. Besides subj-aux inversion, there were transformational accounts of passives, wh-questions, and embedded sentences. The modern descendants of TG do not use transformations in the same way. In **government-binding (GB) theory**, a single transformation rule, Move- \tilde{N} , allows any constituent to be moved anywhere! The focus is on developing constraints on movement that prohibit the generation of illegal surface structures.

the grammar significantly and appears to handle the phenomena quite well. The following sections discuss two different feature-based techniques for context-free grammars and the hold-list technique for ATNs in detail.

[Allen 1995: Chapter 5 - Grammars for Natural Language 132]

BOX 5.2 Different Types of Movement

While the discussion in this chapter will concentrate on wh-questions and thus will examine the movement of wh-terms extensively, the techniques discussed are also needed for other forms of movement as well. Here are some of the most common forms of movement discussed in the linguistics literature. For more details, see a textbook on linguistics, such as Baker (1989).

wh-movement - move a wh-term to the front of the sentence to form a wh-question

topicalization - move a constituent to the beginning of the sentence for emphasis, as in

I never liked this picture.

This picture, I never liked.

adverb preposing - move an adverb to the beginning of the sentence, as in

I will see you tomorrow.

Tomorrow, I will see you.

extraposition - move certain NP complements to the sentence final position, as in

A book discussing evolution was written.

A book was written discussing evolution.

As you consider strategies to handle movement, remember that constituents cannot be moved from any arbitrary position to the front to make a question. For example,

The man who was holding the two balloons will put the box in the corner.

is a well-formed sentence, but you cannot ask the following question, where the gap is indicated by a dash:

***What will the man who was holding - put the box in the corner?**

This is an example of a general constraint on movement out of relative clauses.

>> [back](#)

5.3 Handling Questions in Context-Free Grammars

The goal is to extend a context-free grammar minimally so that it can handle questions. In other words, you want to reuse as much of the original grammar as possible. For yes/no questions, this is easily done. You can extend Grammar 4.7 with one rule that allows an auxiliary before the first NP and handles most examples:

```
S [+inv] -> (AUXAGR ?a SUBCAT ?v) (NP AGR ?a) (VP VFORM ?v)
```

This enforces subject-verb agreement between the AUX and the subject NP, and ensures that the VP has the right VFORM to follow the AUX. This one rule is all

[Allen 1995: Chapter 5 - Grammars for Natural Language 133]

that is needed to handle yes/no questions, and all of the original grammar for assertions can be used directly for yes/no questions.

As mentioned in Section 5.2, a special feature GAP is introduced to handle wh-questions. This feature is passed from mother to subconstituent until the appropriate place for the gap is found in the sentence. At that place, the appropriate constituent can be constructed using no input. This can be done by introducing additional rules with empty right-hand sides. For instance, you might have a rule such as

```
(NP GAP ((CAT NP) (AGR ?a)) AGR ?a) -> ø
```

which builds an NP from no input if the NP sought has a GAP feature that is set to an NP. Furthermore, the AGR feature of this empty NP is set to the AGR feature of the feature that is the value of the GAP. Note that the GAP feature in the mother has another feature structure as its value. To help you identify the structure of such complex values, a smaller font size is used for feature structures acting as values.

You could now write a grammar that passes the GAP feature appropriately. This would be tedious, however, and would also not take advantage of some generalities that seem to hold for the propagation of the GAP feature. Specifically, there seem to be two general ways in which the GAP feature propagates, depending on whether the head constituent is a lexical or nonlexical category. If it is a nonlexical category, the GAP feature is passed from the mother to the head and not to any other subconstituents. For example, a typical S rule with the GAP feature would be

```
(S GAP ?g) -> (NP GAP-) (VP GAP ?g)
```

The GAP can be in the VP, the head subconstituent, but not in the NP subject. For rules with lexical heads, the gap may move to any one of the nonlexical subconstituents. For example, the rule for verbs with a _np_vp:inf

complement,

```
VP -> V[_np_vp:inf] NP PP
```

would result in two rules involving gaps:

```
(VP GAP ?g) -> V[_np_vp:inf] (NP GAP ?g) (PP GAP-)
```

```
(VP GAP ?g) -> V[_np_vp:inf] (NP GAP -) (PP GAP ?g)
```

In other words, the GAP may be in the NP or in the PP, but not both. Setting the GAP feature in all but one subconstituent to `-` guarantees that a gap can be used in only one place.

An algorithm for automatically adding GAP features to a grammar is shown in Figure 5.5. Note that it does not modify any rule that explicitly sets the GAP feature already, allowing the grammar designer to introduce rules that do not follow the conventions encoded in the algorithm. In particular, the rule for subject-aux inversion cannot allow the gap to propagate to the subject NP.

[Allen 1995: Chapter 5 - Grammars for Natural Language 134]

For each rule $Y \rightarrow X_1 \dots H_i \dots X_n$ with head constituent H_i

1. If the rule specifies a GAP feature in some constituent already, then skip.
2. If the head H_i is not a lexical category, then add a GAP feature to the head and the mother, and \neg GAP to the other subconstituents, producing a rule of form:

$(Y \text{ GAP } ?g) \rightarrow (X_1 \text{ GAP } -) \dots (H_i \text{ GAP } ?g) \dots (X_n \text{ GAP } -)$

3. If the head H_i is a lexical category, then for each nonlexical subconstituent X_j , add a rule of the form:

$(Y \text{ GAP } ?g) \rightarrow (X_1 \text{ GAP } -) \dots (X_j \text{ GAP } ?g) \dots (X_n \text{ GAP } -)$

Figure 5.5 An algorithm for adding GAP features to a grammar

Figure 5.5 An algorithm for adding GAP features to a grammar

Using this procedure, a new grammar can be created that handles gaps. All that is left to do is analyze where the fillers for the gaps come from. In whquestions, the fillers are typically NPs or PPs at the start of the sentence and are identified by a new feature WH that identifies a class of phrases that can introduce questions. The WH feature is signaled by words such as *who*, *what*, *when*, *where*, *why*, and *how* (as in *how many* and *how carefully*). These

words fall into several different grammatical categories, as can be seen by considering what type of phrases they replace. In particular, *who*, *whom*, and *what* can appear as pronouns and can be used to specify simple NPs:

Who ate the pizza?

What did you put the book in?

The words "*what*" and "*which*" may appear as determiners in noun phrases, as in

What book did he steal?

Words such as "*where*" and "*when*" appear as prepositional phrases:

Where did you put the book?

The word "*how*" acts as an adverbial modifier to adjective and adverbial phrases:

How quickly did he run?

Finally, the word "*whose*" acts as a possessive pronoun:

Whose book did you find?

The wh-words also can act in different roles. All of the previous examples show their use to introduce wh-questions, which will be indicated by the WH feature value Q. A subset of them can also be used to introduce relative clauses, which will be discussed in Section 5.4. These will also have the **WH feature** value R. To make the WH feature act appropriately, a grammar should satisfy the following constraint: If a phrase contains a

subphrase that has the WH feature, then the larger phrase also has the same WH feature. This way, complex phrases

[Allen 1995: Chapter 5 - Grammars for Natural Language 135]

what:	(CAT PRO WH Q AGR {3s 3p})	when:	(CAT PP-WRD WH {Q R} PFORM TIME)
what:	(CAT QDET WH Q AGR {3s 3p})	who:	(CAT PRO WH {Q R} AGR {3s 3p})
which:	(CAT QDET WH Q AGR {3s 3p})	where:	(CAT PP-WRD WH {Q R} PFORM {LOC MOT})
which:	(CAT PRO WH R AGR {3s 3p})	whose:	(CAT PRO WH {Q R} POSS + AGR {3s 3p})

Figure 5.6 A lexicon for some of the wh-words

Figure 5.6 A lexicon for some of the wh-words

1. $(NP \text{ POSS } ?p \text{ WH } ?w) \rightarrow (PRO \text{ POSS } ?p \text{ WH } ?w)$
2. $(NP \text{ WH } ?w) \rightarrow (DET \text{ WH } ?w \text{ AGR } ?a) (CNP \text{ AGR } ?a)$
3. $CNP \rightarrow N$
4. $CNP \rightarrow ADJ \ N$
5. $DET \rightarrow ART$
6. $(DET \text{ WH } ?w) \rightarrow (NP[+POSS] \text{ WH } ?a)$
7. $(DET \text{ WH } ?w) \rightarrow (QDET \text{ WH } ?w)$
8. $(PP \text{ WH } ?w) \rightarrow P \ (NP \text{ WH } ?w)$
9. $(PP \text{ WH } ?w) \rightarrow (PP\text{-}WRD \text{ WH } ?w)$

Head feature for NP, DET and CNP: AGR

Head feature for PP: PFORM

Grammar 5.7 A simple NP and PP grammar handling wh-words

Grammar 5.7 A simple NP and PP grammar handling wh-words

containing a subconstituent with a WH word also can be used in questions. For example, the sentence

In what store did you buy the picture?

is a legal question since the initial PP in what store has the WH value Q because the NP what store has the WH value Q (because the determiner what has the WH value Q). With this constraint, the final S constituent will also have the WH value Q, which will be used to indicate that the sentence is a wh-question. Figure 5.6 gives some lexical entries for some wh-words. Note the introduction of new lexical categories: QDET for determiners that introduce wh-terms, and PP-WRD for words like when that act like prepositional phrases. Grammar 5.7 gives some rules for NPs and PPs that handle the WH feature appropriately.

[Allen 1995: Chapter 5 - Grammars for Natural Language 136]

10. $(S[-\text{inv}] \text{ WH } ?w) \rightarrow$
 $(\text{NP WH } ?w \text{ AGR } ?a)$
 $(\text{VP } [fin] \text{ AGR } ?a)$
11. $(S[+\text{inv}] \text{ WH } ?w \text{ GAP } ?g) \rightarrow$
 $(\text{AUX COMPFORM } ?s \text{ AGR } ?a)$
 $(\text{NP WH } ?w \text{ AGR } ?a \text{ GAP } \neg)$
 $(\text{VP VFORM } ?s \text{ GAP } ?g)$
12. $S \rightarrow (\text{NP}[Q, -\text{gap}] \text{ AGR } ?a) (S[+\text{inv}] \text{ GAP } (\text{NP AGR } ?a))$
13. $S \rightarrow (\text{PP}[Q, -\text{gap}] \text{ PFORM } ?p) (S[+\text{inv}] \text{ GAP } (\text{PP PFORM } ?p))$
14. $\text{VP} \rightarrow (\text{AUX COMPFORM } ?s) (\text{VP VFORM } ?s)$
15. $\text{VP} \rightarrow V[_{\text{none}}]$
16. $\text{VP} \rightarrow V[_{\text{np}}] \text{ NP}$
17. $\text{VP} \rightarrow V[_{\text{vp:inf}}] \text{ VP[inf]}$
18. $\text{VP} \rightarrow V[_{\text{np_vp:inf}}] \text{ NP VP[inf]}$
19. $\text{VP[inf]} \rightarrow TO \text{ VP[base]}$
20. $\text{VP} \rightarrow V[_{\text{np_pp:loc}}] \text{ NP PP[loc]}$

Head features for S, VP: VFORM, AGR

Grammar 5.8 The unexpanded S grammar for wh-questions

Grammar 5.8 The unexpanded S grammar for wh-questions

With the WH feature defined and a mechanism for modifying a grammar to handle the GAP feature, most wh-questions can be handled by adding only two new rules to the grammar. Here are the rules for NP- and PP-based questions:

```
S -> (NP[Q,-gap] AGR ?a) (S[+inv] GAP (NP AGR ?a))  
S -> (PP[Q,-gap] PFORM ?p) (S[+inv] GAP (PP PFORM ?p))
```

Both of these rules set the value of the GAP feature to a copy of the initial WH constituent so that it will be used to fill a gap in the S constituent. Since the S constituent must have the feature +INV, it must also contain a subject-aux inversion. Given the NP and PP rules in Grammar 5.7, Grammar 5.8 provides the S and VP rules to handle questions. Grammar 5.9 shows all the derived rules introduced by the procedure that adds the GAP features. Rules 11, 12, and 13 are not changed by the algorithm as they already specify the GAP feature.

Parsing with Gaps

A grammar that allows gaps creates some new complications for parsing algorithms. In particular, rules that have an empty right-hand side, such as

```
(NP AGR ?a GAP (NP AGR ?a)) -> ø
```

[Allen 1995: Chapter 5 - Grammars for Natural Language 137]

10. $(S[-\text{inv}] \text{ WH } ?w \text{ GAP } ?g) \rightarrow$
 $(\text{NP WH } ?w \text{ AGR } ?a)$
 $(\text{VP}[\text{fin}] \text{ AGR } ?a \text{ GAP } ?g)$
11. $(S[+\text{inv}] \text{ WH } ?w \text{ GAP } ?g) \rightarrow$
 $(\text{AUX COMPFORM } ?s \text{ AGR } ?a)$
 $(\text{NP WH } ?w \text{ AGR } ?a \text{ GAP } -)$
 $(\text{VP VFORM } ?s \text{ GAP } ?g)$
12. $S \rightarrow (\text{NP}[Q, -\text{gap}] \text{ AGR } ?a) (S[+\text{inv}] \text{ GAP } (\text{NP AGR } ?a))$
13. $S \rightarrow (\text{PP}[Q, -\text{gap}] \text{ PFORM } ?p) (S[+\text{inv}] \text{ GAP } (\text{PP PFORM } ?p))$
14. $(\text{VP GAP } ?g) \rightarrow (\text{AUX COMPFORM } ?s) (\text{VP VFORM } ?s \text{ GAP } ?g)$
15. $\text{VP} \rightarrow V[\text{none}]$
16. $(\text{VP GAP } ?g) \rightarrow V[\text{np}] (\text{NP GAP } ?g)$
17. $(\text{VP GAP } ?g) \rightarrow V[\text{vp:inf}] (\text{VP[inf]} \text{ GAP } ?g)$
18. $(\text{VP GAP } ?g) \rightarrow V[\text{np_vp:inf}] (\text{NP GAP } ?g) (\text{VP[inf]} \text{ GAP } -)$
- 18'. $(\text{VP GAP } ?g) \rightarrow V[\text{np_vp:inf}] (\text{NP GAP } -) (\text{VP[inf]} \text{ GAP } ?g)$
19. $(\text{VP[inf]} \text{ GAP } ?g) \rightarrow TO (\text{VP[base]} \text{ GAP } ?g)$
20. $(\text{VP GAP } ?g) \rightarrow V[\text{np_pp:loc}] (\text{NP GAP } ?g) (\text{PP[loc]} \text{ GAP } -)$
- 20'. $((\text{VP GAP } ?g) \rightarrow V[\text{np_pp:loc}] (\text{NP GAP } -) (\text{PP[loc]} \text{ GAP } ?g))$

Head features for S, VP: VFORM, AGR

Grammar 5.9 The S grammar for wh-questions with the GAP feature

Grammar 5.9 The S grammar for wh-questions with the GAP feature

may cause problems because they allow an empty NP constituent anywhere. In a bottom-up strategy, for instance, this rule could apply at any position (or many times at any position) to create NPs that use no input. A top-down strategy fares better, in that the rule would only be used when a gap is explicitly predicted. Instead of using such rules, however, the arc extension algorithm can be modified to handle gaps automatically. This technique works with any parsing strategy.

Consider what extensions are necessary. When a constituent has a GAP feature that matches the constituent itself, it must be realized by the empty constituent. This means that an arc whose next constituent is a gap can be extended immediately. For example, consider what happens if the following arc is suggested by the parser:

```
(VP GAP (NP AGR 3s))  
-> V[_np_pp:loc] o (NP GAP (NP AGR 3s)) PP[LOC]
```

The next constituent needed is an NP, but it also has a GAP feature that must be an NP. Thus the constituent must be empty. The parser inserts the constituent

```
(NP AGR 3s EMPTY +)
```

[Allen 1995: Chapter 5 - Grammars for Natural Language 138]

BOX 5.3 The Movement Constraints

In linguistics the principles that govern where gaps may occur are called island constraints. The term draws on the metaphor of constituent movement. An island is a constituent from which no subconstituent can move out (just as a person cannot walk off an island). Here are a few of the constraints that have been proposed.

The A over A Constraint - No constituent of category A can be moved out of a constituent of type A. This means you cannot have an NP gap within an NP, a PP gap within a PP, and so on, and provides justification for not allowing non-null constituents of the form NP/NP, PP/PP, and so on. This disallows sentences such as

* What book₁ did you meet the author of - ₁?

Complex-NP Constraint - No constituent may be moved out of a relative clause or noun complement. This constraint disallows sentences like

* To whom₁ did the man who gave the book –₁ laughed?

where the PP *to whom* would have been part of the relative clause *who gave the book to whom* (as in *The man who gave the book to John laughed*).

Sentential Subject Constraint - No constituent can be moved out of a constituent serving as the subject of a sentence. This overlaps with the other constraints when the subject is an NP, but non-NP subjects are possible as well, as in the sentence *For me to learn these constraints is impossible*. This constraint eliminates the possibility of a question like

What₁ is for me to learn –₁ impossible?

Wh-Island Constraint - No constituent can be moved from an embedded sentence with a wh-complementizer. For example, while *Did they wonder whether I took the book?* is an acceptable sentence, you cannot ask

*What₁ did they wonder whether I took –₁?

Coordinate Structure Constraint - A constituent cannot be moved out of a coordinate structure. For example, while *Did you see John and Sam?* is an acceptable sentence, you cannot ask

*Who₁ did you see John and –₁?

Note that these constraints apply to all forms of movement, not just wh-questions. For example, they constrain topicalization and adverb preposing as well.

to the chart, which can then extend the arc to produce the new arc

```
(VP GAP (NP AGR 3s))  
-> V[_np_pp:loc] (NP GAP (NP AGR 3s)) o PP(LOC)
```

A more precise specification of the algorithm is given in Figure 5.10.

[Allen 1995: Chapter 5 - Grammars for Natural Language 139]

Whenever an arc of the form

$X \rightarrow \dots \circ (C F_1 V_1 \dots F_n V_n \text{ GAP } (C G_1 ?v_{g1} \dots G_m ?v_{gm})) \dots$

is suggested by the parser, and the constituent pattern that is the GAP feature, that is,

$(C G_1 ?v_{g1} \dots G_m ?v_{gm})$

matches the constituent itself

$(C F_1 V_1 \dots F_n V_n \text{ GAP } (C G_1 V_{G1} \dots G_m V_{Gm}))$

then add a new constituent $(C G_1 ?v_{g1} \dots G_m ?v_{gm} \text{ EMPTY } +)$, with the variables bound as necessary, to the chart. Use this constituent to extend the original arc.

Figure 5.10 The algorithm to insert empty constituents as needed

Figure 5.10 The algorithm to insert empty constituents as needed

Consider parsing "*Which dogs did he see?*" using the bottom-up strategy. Only the rules that contribute to the final analysis will be considered. Rule 7 in Grammar 5.7 applies to the constituent QDET1 (*which*) to build a constituent DET1, and rule 3 applies to the constituent N1 (*dogs*) to build a constituent CNP1, which then is combined with DET1 by rule 2 to build an NP constituent of the form

NP1: (NP AGR 3p

WH Q

1 QDET1

2 CNP1)

This NP introduces an arc based on rule 12 in Grammar 5.8. The next word, "did", is an AUX constituent, which introduces an arc based on rule 11. The next word, "he", is a PRO, which creates an NP, NP2, using rule 1, and extends the arc based on rule 1. The chart at this stage is shown in Figure 5.11.

The word "see" introduces a verb V1, which can extend rule 16. This adds the arc labeled

(VP GAP ?g) -> V[_np] o (NP GAP ?g)

Since the GAP value can match the required NP (because it is unconstrained), an empty NP is inserted in the chart with the form

EMPTY-NP1: (NP AGR ?a GAP (NP AGR ?a)) EMPTY +)

This constituent can then be used to extend the VP arc, producing the constituent

```
VP1: (VP VF0RM inf  
      GAP (NP AGR ?a)  
      1 V1  
      2 EMPTY-NP1)
```

Now VP1 can extend the arc based on rule 11 to form the S structure:

[Allen 1995: Chapter 5 - Grammars for Natural Language 140]

NP1			
WH Q AGR 3p 1 DET1 2 CNP1			
DET1 WH Q AGR 3p 1 QDET1	CNP1 AGR 3p 1 N1		NP2 AGR 3s 1 PRO1
QDET1 WH Q AGR 3p	N1 AGR 3p	AUX1 AGR 3s VFORM past SUBCAT base	PRO1 AGR 3s

Which dogs did he

$S \rightarrow NP[Q] \circ (S \text{ GAP } (NP \text{ AGR } 3p))$

$(S \text{ GAP } ?g) \rightarrow AUX \text{ NP } \circ (VP \text{ GAP } ?g)$

Figure 5.11 The chart after the word *he*

Figure 5.11 The chart after the word he

S1: (S GAP(NP AGR ?a)

INV +

1 AUX1

2 NP2

3 VP1)

Finally, NP1 and S1 can combine using rule 12 to produce an S constituent. The final chart is shown in Figure 5.12.

Thus, by using the WH and GAP features appropriately and by extending the parser to insert empty constituents as needed to fill gaps, the original grammar for declarative assertions can be extended to handle yes/no questions and wh-questions with only three new rules. The success of this approach depends on getting the right constraints on the propagation of the GAP feature. For instance, in a rule a gap should only be passed from the mother to

exactly one of the subconstituents. If this constraint were not enforced, many illegal sentences would be allowed. There are some cases where this constraint appears to be violated. They are discussed in Exercise 6.

With a bottom-up parsing strategy, many empty constituents are inserted by the parser, and many constants with gaps are added to the chart. Most of these are not used in any valid analysis of the sentence. The top-down strategy greatly reduces the number of empty constants proposed and results in a considerably smaller chart.

>> [back](#)

[Allen 1995: Chapter 5 - Grammars for Natural Language 141]

S2				
VFORM past 1 NP1 2 S1				
NP1		S1		
WH Q AGR 3p 1 DET1 2 CNP1		INV+ GAP (NP AGR 3p) VFORM past 1 AUX1 2 NP2 3 VP1		
DET1	CNP1		NP2	VP1
WH Q AGR 3p 1 QDET1	AGR 3p 1 N1		AGR 3s 1 PRO1	VFORM inf GAP (NP AGR 3p) 1 V1 2 EMPTY-NP1
QDET1	N1	AUX1 AGR 3s VFORM past SUBCAT base	PRO1 AGR 3s	

Which dogs did he see

Figure 5.12 The final chart for *Which dogs did he see?*

Figure 5.12 The final chart for Which dogs did he see?

5.4 Relative Clauses

This section examines the structure of relative clauses. A relative clause can be introduced with a rule using a new category, REL:

```
CNP -> CNP REL
```

A large class of relative clauses can be handled using the existing grammar for questions as they involve similar constructs of a wh-word followed by an S structure with a gap. The main difference is that only certain wh-words are allowed (*who*, *whom*, *which*, *when*, *where*, and *whose*) and the S-structure is not inverted. The special class of wh-words for relative clauses is indicated using the WH feature value R. Figure 5.6 showed lexicon entries for some of these words.

Given this analysis, the following rules

```
REL -> (NP WH R AGR ?a) (S[-inv, fin] GAP (NP AGR ?a))  
REL -> (PP WH R PFORM ?p) (S[-inv, fin] GAP (PP PFORM ?p))
```

handle relative clauses such as

The man who we saw at the store

The exam in which you found the error

The man whose book you stole

[Allen 1995: Chapter 5 - Grammars for Natural Language 142]

BOX 5.4 Feature Propagation In GPSG

GPSG (generalized phrase structure grammar) (Gazdar et al., 1985) formalizes all feature propagation in terms of a set of general principles. The GAP feature corresponds to the SLASH feature in GPSG. Their claim is that no special mechanism is needed to handle the SLASH feature beyond what is necessary for other features. In particular, the SLASH feature obeys two general GPSG principles. The head feature principle, already discussed, requires all head features to be shared between a head subconstituent and its mother constituent. The foot feature principle requires that if a foot feature appears in any subconstituent, it must be shared with the mother constituent. The constraints on the propagation of the SLASH feature result from it being both a head feature and a foot feature.

GPSG uses meta-rules to derive additional rules from a basic grammar. Meta-rules only apply to rules with lexical heads. One meta-rule accounts for gaps by taking an initial rule, for example,

VP → V NP

and producing a new rule

VP/NP → V NP [+NULL]

where VP/NP is a VP with the SLASH feature NP, and the feature +NULL indicates that the constituent is "phonologically null", that is, not stated or present in the sentence.

GPSG meta-rules are used for other phenomena as well. For example, one meta-rule generates rules for passive voice sentences from rules for active voice sentences, while another generates subject-aux inversion rules from noninverted S rules. Meta-rules play much the same role as transformations, but they are limited to act on only a single rule at a time, and that rule must have a lexical head, whereas transformations may operate on arbitrary trees. The passive meta-rule looks something like this:

VP → V[TRANSITIVE] NP X =>

VP[PASSGAP] → V X

Here the symbol X stands for any sequence of constituents. Thus this meta-rule says that if there is a rule for a VP consisting of a transitive verb, an NP, and possibly some other constituents, then there is another rule for a passive VP consisting of the same verb and constituents, except for the NP. As long as there is an upper

limit to the number of symbols the variable X can match, you can show that the language described by the grammar expanded by all possible mets-rule applications remains a context-free language.

Because gaps don't propagate into the subject of a sentence, a new rule is needed to allow NPs like "*The man who read the paper*", where the wh-word plays the role of subject. This can be covered by the rule

[Allen 1995: Chapter 5 - Grammars for Natural Language 143]

REL → NP [R] VP[fin]

In addition, there is a common class of relative clauses that consist of "*that*" followed by an S with an NP gap, or a VP, as in "*The man that we saw at the party*" and "*The man that read the paper*". If you allow *that* to be a relative pronoun with WH feature R, then the above rules also cover these cases. Otherwise, additional rules must be added for "*that*".

Finally, there are relative clauses that do not start with an appropriate whphrase but otherwise look like normal relative clauses. Examples are

The paper John read

The damage caused by the storm

The issue creating the argument

The latter two examples involve what are often called reduced relative clauses. These require two additional rules allowing a relative clause to be a finite S with an NP gap or a VP in a participle form:

REL -> (S[fin] GAP (NP AGR ?a))

REL -> (VP VFORM {ing pastprt})

Notice that now there are two major uses of gaps: one to handle questions and the other to handle relative clauses. Because of this, you should be careful to examine possible interactions between the two. What happens to relative clauses within questions? They certainly can occur, as in

Which dog₁ did the man who₂ we saw –₂ holding the bone feed –₁?

The gaps are shown using dashes, and numbers indicate the fillers for the gaps. The existing grammar does allow this sentence, but it is not obvious how, since the GAP feature can only store one constituent at a time and the sentence seems to require storing two. The answer to this puzzle is that the way the GAP feature is propagated

does not allow the initial Q constituent to move into the relative clause. This is because REL is not the head of the rule. In particular, the rule

CNP → CNP REL

expands to the rule

(CNP GAP ?g) → (CNP GAP ?g) (REL GAP-)

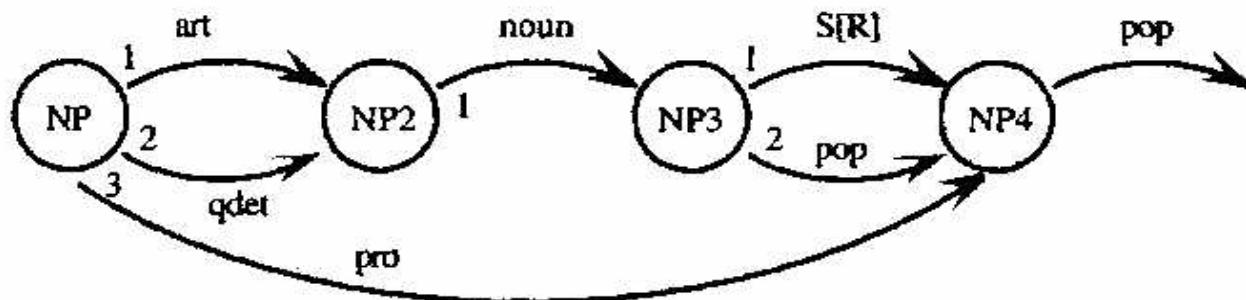
Thus the gap for the question cannot be found in the relative clause. As a result, when a new gap is proposed within the relative clause, there is no problem.

So the mechanism works in this case, but does it capture the structure of English well? In particular, can a question gap ever appear within a relative clause? The generally accepted answer appears to be no. Otherwise, a question like the following would be acceptable:

*** Which dog₁ did the man₂ we saw –₂ petting –₁ laughed?**

So the mechanism proposed here appears to capture the phenomena well.

[Allen 1995: Chapter 5 - Grammars for Natural Language 144]



Arc	Test	Actions
NP/1	-----	DET := * AGR := AGR *
NP/2	-----	DET := * WH := Q AGR := AGR *
NP/3	-----	PRO := * WH := WH * AGR := AGR *
NP2/1	AGR \cap AGR *	HEAD := * AGR := AGR \cap AGR *
NP3/1	WH * \cap R	MOD := *

Grammar 5.13 An NP network including wh-words

Grammar 5.13 An NP network including wh-words

>> [back](#)

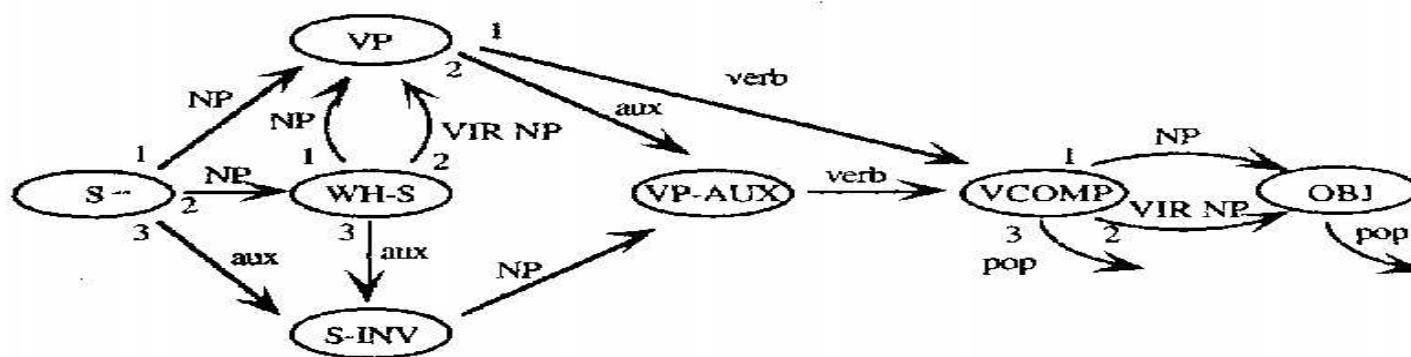
5.5 The Hold Mechanism in ATNs

Another technique for handling movement was first developed with the ATN framework. A data structure called the hold list maintains the constituents that are to be moved. Unlike GAP features, more than one constituent may be on the hold list at a single time. Constituents are added to the hold list by a new action on arcs, the hold action, which takes a constituent and places it on the hold list.

The hold action can store a constituent currently in a register (for example, the action HOLD SUBJ holds the constituent that is in the SUBJ register). To ensure that a held constituent is always used to fill a gap, an ATN system does not allow a pop arc to succeed from a network until any constituent held by an action on an arc in that network has been used. That is, the held constituent must have been used to fill a gap in the current constituent or in one of its subconstituents.

Finally, you need a mechanism to detect and fill gaps. A new arc called VIR (for virtual) that takes a constituent name as an argument can be followed if a constituent of the named category is present on the hold list. If the arc is followed successfully, the constituent is removed from the hold list and returned as the value of the arc in the identical form that a PUSH arc returns a constituent.

[Allen 1995: Chapter 5 - Grammars for Natural Language 145]



Arc	Test	Actions
S/1		INV := - SUBJ := *
S/2	$WH_* \cap \{Q, R\}$	$WH := WH^*$ HOLD *
S/3	$VFORM_* \cap \{\text{past pres}\}$	INV := + AUX := * SUBJ := *
WH-S/1		SUBJ := *
WH-S/2		SUBJ := *
WH-S/3	$VFORM_* \cap \{\text{past pres}\}$	AUX := *
VP/1	$AGR_{SUBJ} \cap AGR_*$ $VFORM_* \cap \{\text{past pres}\}$	MAIN-V := *
VP/2	$AGR_{SUBJ} \cap AGR_*$ $VFORM_* \cap \{\text{past pres}\}$	AUX := *
S-INV/1	$AGR_{AUX} \cap AGR_*$	SUBJ := *
VP-AUX/1	$SUBCAT_{AUX} \cap FORM_*$	MAIN-V := *
VCOMP/1	$SUBCAT_{MAIN-V} \cap _\text{np}$	OBJ := *
VCOMP/2	$SUBCAT_{MAIN-V} \cap _\text{np}$	OBJ := *
VCOMP/3	$SUBCAT_{MAIN-V} \cap _\text{none}$	

Grammar 5.14 An S network for questions and relative clauses

Grammar 5.14 An S network for questions and relative clauses

Grammar 5.13 shows an NP network that recognizes NPs involving whwords as pronouns or determiners. As with CFGs, the feature WH is set to the value Q to indicate the NP is a wh-NP starting a question. It also accepts relative clauses with a push to the S network, shown in Grammar 5.14. This S network handles yes/no questions, wh-questions, and relative clauses. Wh-questions and relative clauses are processed by putting the initial NP (with a WH feature, Q, or

[Allen 1995: Chapter 5 - Grammars for Natural Language 146]

R) onto the hold list, and later using it in a VIR arc. The network is organized so that all +INV sentences go through node S-INV, while all -INV sentences go through node VP. All wh-questions and relative clauses go through node WH-S and then are redirected back into the standard network based on whether or not the sentence

is inverted. The initial NP is put on the hold list when arc Sf2 is followed. For noninverted questions, such as *Who ate the pizza?*, and for relative clauses in the NP of form *the man who ate the pizza*, the held NP is immediately used by the VIR arc WH-S/2. For other relative clauses, as in the NP *the man who we saw*, arc WH-S/1 is used to accept the subject in the relative clause, and the held NP is used later on arc VCOMP/2. For inverted questions, such as *Who do I see?*, arc WH-S/3 is followed to accept the auxiliary, and the held NP must be used later.

This network only accepts verbs with SUBCAT values *_none* and *_np* but could easily be extended to handle other verb complements. When extended, there would be a much wider range of locations where the held NP might be used. Figure 5.15 shows a trace of the sentence "*The man who we saw cried*". In parsing the relative clause, the relative pronoun *who* is held in step 5 and used by a VIR arc in step 8.

Note that this ATN would not accept **Who did the man see the boy?*, as the held constituent *who* is not used by any VIR arc; thus the pop arc from the S network cannot be taken. Similarly, **The man who the boy cried ate the pie* is unacceptable, as the relative pronoun is not used by a VIR arc in the S network that analyzes the relative clause.

Comparing the Methods

You have seen two approaches to handling questions in grammars: the use of the GAP feature and the use of the hold list in ATNs. To decide which is best, we must determine the criteria by which to evaluate them. There are three important considerations: coverage - whether the approach can handle all examples; selectivity - whether it rejects all ill-formed examples; and conciseness - how easily rules can be specified.

Under reasonable assumptions both methods appear to have the necessary coverage, that is, a grammar can be written with either approach that accepts any acceptable sentence. Given that this is the case, the important issues become selectivity and conciseness, and here there are some differences. For instance, one of the underlying assumptions of the GAP feature theory is that a symbol such as NP with an NP gap must be realized as the empty string; that is, you could not have an NP with an NP gap inside it. Such a structure could be parsable using the ATN hold list mechanism. In particular, a sentence such as **Who did the man who saw hit the boy?*, while not comprehensible, would be accepted by the ATN in Grammars 5.13 and 5.14 because the hold list would contain two NPs; one for each occurrence of *who*. In the relative clause, the relative pronoun would be taken as the subject and the initial query NP would be taken as the

[Allen 1995: Chapter 5 - Grammars for Natural Language 147]

Trace of S Network

Step	Node	Position	Arc Followed
1.	S	1	S/1 (for recursive call see trace below)
11.	VP	6	VP/1
12.	VCOMP	7	VCOMP/3 succeeds since no words left

Registers

SUBJ \leftarrow (NP DET the
HEAD man
AGR 3s
MOD (S who we saw))

MAIN-V \leftarrow cried

returns

(S SUBJ (NP DET the
HEAD man
AGR 3p
MOD (S who we saw))
MAIN-V cried)

Trace of First NP Call: Arc S/1

Step	Node	Position	Arc Followed
2.	NP	1	NP/1
3.	NP2	2	NP2/1
4.	NP3	3	NP3/1 (for recursive call see trace below)
10.	NP4	6	NP4/1 pop

Registers

DET \leftarrow the
AGR \leftarrow {3s 3p}

HEAD \leftarrow man

AGR \leftarrow 3s

MOD \leftarrow (S WH R
SUBJ we
MAIN-V saw
OBJ who)

returns (NP DET the
HEAD man
AGR 3s
MOD (S who we saw))

Trace of Recursive Call to S on Arc NP3/1

Step	Node	Position	Arc Followed
5.	S	3	S/2 (call to NP network not shown)
6.	WH-S	4	WH-S/1
7.	VP	5	VP/1
8.	VCOMP	6	VCOMP/2 (uses the NP on the hold list)
9.	OBJ	6	OBJ/1 pop

Registers

WH \leftarrow {Q R}
HOLDING (NP PRO who
WH (Q R))

WH \leftarrow R

SUBJ \leftarrow (NP PRO we ...)

MAIN-V \leftarrow saw

OBJ \leftarrow (NP PRO who ...)

returns (S WH R
SUBJ we
MAIN-V saw
OBJ who)

Figure 5.15 A trace of an ATN parse for *The_2 man_3 who_4 we_5 saw_6 cried_7*

Figure 5.15 A trace of an ATN parse for ₁ The ₂man ₃who₄we ₅saw ₆cried ₇

object. This sentence is not acceptable by any grammar using GAP features, however, because the GAP feature cannot be propagated into a relative clause since it is not the head constituent in the rule

[Allen 1995: Chapter 5 - Grammars for Natural Language 148]

CNP → CNP REL

Even if it were the head, however, the sentence would still not be acceptable, since the erroneous analysis would require the GAP feature to have two values when it starts analyzing the relative clause.

The hold list mechanism in the ATN framework must be extended to capture these constraints, because there is no obvious way to prevent held constituents from being used anytime they are available. The only possible way to restrict it using the existing mechanism would be to use feature values to keep track of what held constituents are available in each context. This could become quite messy. You can, however, make a simple extension. You can introduce a new action - say, HIDE - that temporarily hides the existing constituents on the hold list until either an explicit action - say, UNHIDE - is executed, or the present constituent is completed. With this extension the ATN grammar to handle relative clauses could be modified to execute a HIDE action just before the hold action that holds the current constituent is performed.

In this case, however, it is not the formalism itself that embodies the constraints. Rather, the formalism is used to state the constraints; it could just as easily describe a language that violates the constraints. The theory based on GAP features, on the other hand, embodies a set of constraints in its definition of how features can propagate. As a result, it would be much more difficult to write a grammar that describes a language that violates the movement constraints.

While a true account of all the movement constraints is still an area of research, this simple analysis makes an important point. If a formalism is too weak, you cannot describe the language at all. If it is too strong, the grammar may become overly complicated in order to eliminate sentence structures that it can describe but that are not possible. The best solution, then, is a formalism that is just powerful enough to describe natural languages. In such a language many of the constraints that first appear to be arbitrary restrictions in a language might turn out to be a consequence of the formalism itself and need no further consideration.

While the technique of GAP feature propagation was introduced with context-free grammars, and the hold list mechanism was introduced with ATNs, these techniques are not necessarily restricted to these formalisms. For instance, you could develop an extension to CFGs that incorporates a hold list and uses it for gaps. Likewise, you

might be able to develop some rules for GAP feature propagation in an ATN. This would be more difficult, however, since the GAP feature propagation rules depend on the notion of a head subconstituent, which doesn't have a direct correlate in ATN grammars.

>> [back](#)

5.6 Gap Threading

A third method for handling gaps combines aspects of both the GAP feature approach and the hold list approach. This technique is usually called gap threading. It is often used in logic grammars, where two extra argument

[Allen 1995: Chapter 5 - Grammars for Natural Language 149]

1. $s(\text{In}, \text{Out}, \text{FillersIn}, \text{FillersOut}) :- np(\text{In}, \text{In1}, \text{FillersIn}, \text{Fillers1}),$
 $\quad vp(\text{In1}, \text{Out}, \text{Fillers1}, \text{FillersOut})$
2. $vp(\text{In}, \text{Out}, \text{FillersIn}, \text{FillersOut}) :- v(\text{In}, \text{In1})$
3. $vp(\text{In}, \text{Out}, \text{FillersIn}, \text{FillersOut}) :- v(\text{In}, \text{In1}), np(\text{In1}, \text{Out}, \text{FillersIn}, \text{FillersOut})$
4. $np(\text{In}, \text{Out}, \text{Fillers}, \text{Fillers}) :- art(\text{In}, \text{In1}), cnp(\text{In1}, \text{Out})$
5. $np(\text{In}, \text{Out}, \text{Fillers}, \text{Fillers}) :- pro(\text{In}, \text{Out})$
6. $cnp(\text{In}, \text{Out}) :- n(\text{In}, \text{In1}), np\text{-comp}(\text{In1}, \text{Out})$
7. $np\text{-comp}(\text{In}, \text{In}) :-$
(This covers the case where there is no NP complement.)
8. $np\text{-comp}(\text{In}, \text{Out}) :- rel\text{-intro}(\text{In}, \text{In1}, \text{Filler}),$
 $\quad s(\text{In1}, \text{Out}, (\text{Filler } \text{nil}), \text{nil})$
(Here we hold the Rel-Intro constituent, and must use it in the following S.)
9. $rel\text{-intro}(\text{In}, \text{Out}, [\text{NP}]) :- relpro(\text{In}, \text{Out})$
(where relpro accepts any pronoun with WH feature R)
10. $np(\text{In}, \text{In}, [\text{NP} \mid \text{Fillers}], \text{Fillers}) :-$
(This rule builds an empty np from a filler.)

Grammar 5.16 A logic grammar using gap threading

Grammar 5.16 A logic grammar using gap threading

positions are added to each predicate—one argument for a list of fillers that might be used in the current constituent, and one for the resulting list of fillers that were not used after the constituent is parsed. Thus the predicate

s (position-in, position-out, fillers-in, fillers-out)

is true only if there is a legal S constituent between *position-in* and *position-out* of the input. If a gap was used to build the S, its filler will be present in *fillers-in*, but not in *fillers-out*. For example, an S constituent with an NP gap would correspond to the predicate s([n, Out, (NP], nil). In cases where there are no gaps in a constituent, the *fillers-in* and *fillers-out* will be identical.

Consider an example dealing with relative clauses. The rules required are shown in Grammar 5.16. The various feature restrictions that would be needed to enforce agreement and subcategorization are not shown so as to keep the example simple.

To see these rules in use, consider the parse of the sentence *The man who we saw cried* in Figure 5.17. The relative clause is analyzed starting with step 7. Using rule 9, the word *who* is recognized as a relative pronoun, and the variable Filler is bound to the list [NP]. This filler is then passed into the embedded S (step 9), to the NP (step 10), and then on to the VP (step 12), since it is not used in the NP. From there it is passed to the NP predicate in step 14, which uses the filler according to rule 10. Note that no other NP rule could have applied at this point, because the filler must be used since the FillersOut variable is nil. Only rules that consume the filler can apply. Once this gap is used, the entire NP from positions 1 to 6 has been found and the rest of the parse is straightforward.

[Allen 1995: Chapter 5 - Grammars for Natural Language 150]



Figure 5.17 A trace of the parse of $_1 \text{The} _2 \text{man} _3 \text{who} _4 \text{we} _5 \text{saw} _6 \text{cried} _7$

Just as a convenient notation was designed for definite clause grammars, which then could be simply translated into a PROLOG program, a notation has been designed to facilitate the specification of grammars that handle gaps. In particular, there is a formalism called extraposition grammar, which, besides allowing normal context-free rules, allows rules of the form

REL-MARK . . . TRACE -> REL-PRO

which essentially says that the constituent REL-MARK, plus the constituent TRACE later in the sentence, can be rewritten as a REL-PRO. Such a rule violates the tree structure of syntactic forms and allows the analysis shown in Figure 5.18 of the NP *the mouse that the cat ate*. Such rules can be compiled into a brief rrammar usin2 the gap-threading technique.

[Allen 1995: Chapter 5 - Grammars for Natural Language 151]

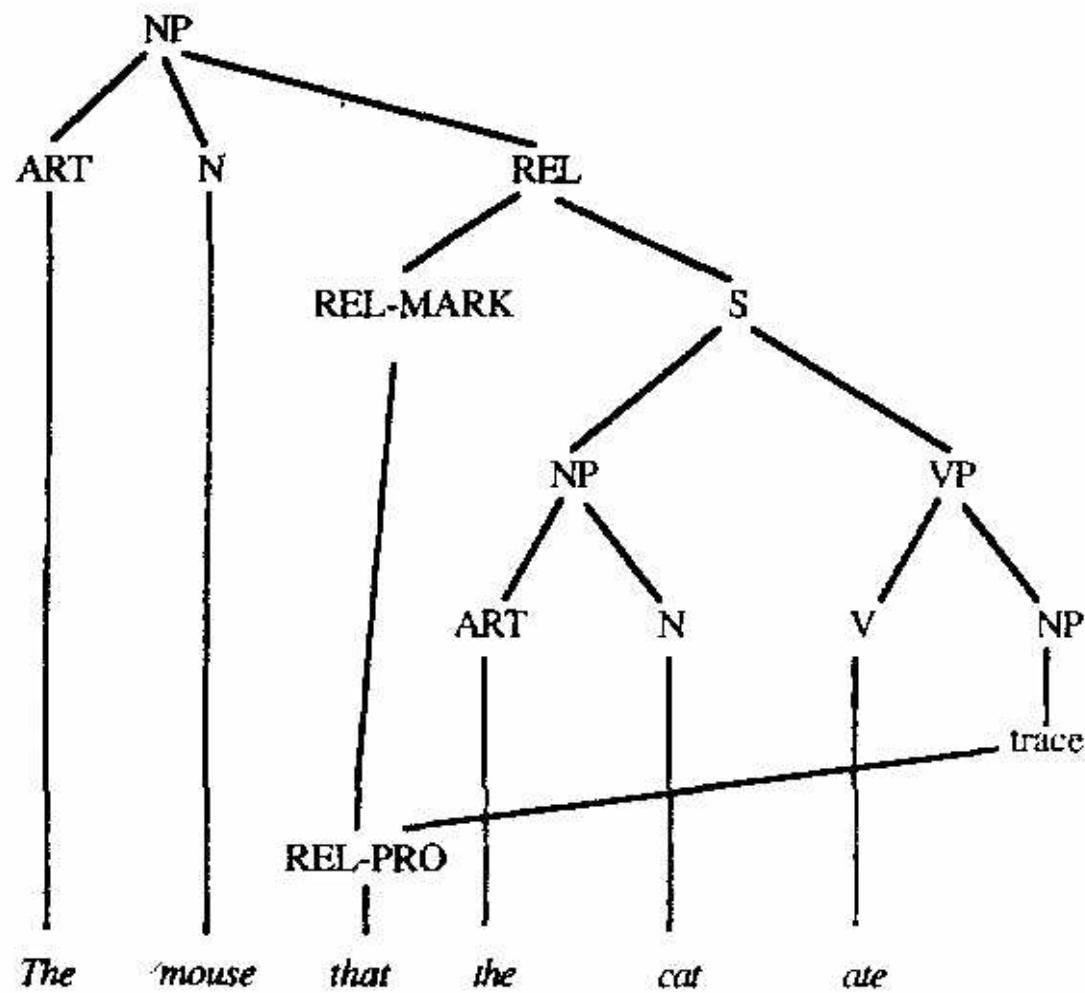


Figure 5.18 A parse tree in extraposition grammar

Figure 5.18 A parse tree in extraposition grammar

Consider how gap threading compares with the other approaches. Of course, by using additional arguments on predicates, any of the approaches could be implemented. If viewed as simply an implementation technique, then the gap-threading approach looks like a way to implement a hold list in a logic grammar. Since the grammar designer can decide to propagate the hold list or not in the grammar, it provides the flexibility to avoid some of the problems with the simple hold list mechanism. For instance, Grammar 5.16 does not pass the fillers from outside a noun phrase into its relative clause. So problems like those discussed in the last section can be avoided. It also would be possible to adopt the GAP feature introduction algorithm described in Section 5.3 so that it introduces gap-threading rules into a logic grammar. Thus, the approach provides the grammar writer with great flexibility. Like the hold list approach, however, the propagation constraints must be explicitly enforced rather than being a consequence of the formalism.

>> [back](#)

Summary

Many of the complex aspects of natural language can be treated as movement phenomena, where a constituent in one position is used to satisfy the constraints at another position. These phenomena are divided into two classes: bounded movement, including yes/no questions and passives; and unbounded movement, including wh-questions and relative clauses. The computational techniques developed for handling movement allow significant generalities to be captured

[Allen 1995: Chapter 5 - Grammars for Natural Language 152]

between all of these different sentential forms. By using the feature system carefully, a basic grammar of declarative sentences can be extended to handle the other forms with only a few rules. Three different techniques

were introduced to handle such phenomena. The first was the use of the special feature propagation rules using a feature called GAP. The second was the hold list mechanism in ATNs, which involves adding a new action (the hold action) and a new, arc type (the V1R arc). The third involved gap threading, with a hold-list-like structure passed from constituent to constituent using features. With suitable care, all of these approaches have been used successfully to build grammars of English with substantial coverage.

>> [back](#)

Related Work and Further Readings

The phenomena of unbounded dependencies has motivated a significant amount of research into grammatical formalisms. These fall roughly into several categories, depending on how close the grammar remains to the context-free grammars. Theories such as transformational grammar (Chomsky, 1965; Radford, 1981), for example, propose to handle unbounded dependencies completely outside the CFG framework. Theories such as lexical functional grammar (Kaplan and Bresnan, 1982) and generalized phrase structure grammar (GPSG) (Gazdar, 1982; Gazdar et al., 1985) propose methods of capturing the dependencies with the CFG formalism using features. There has also been considerable work in defining new formalisms that are slightly more powerful than context-free grammars and can handle long-distance dependencies such as tree -adjoining grammars (TAGs) (see Box 5.5) and combinatory categorial grammars (Steedman, 1987).

The first reasonably comprehensive computational study of movement phenomena was in the ATN framework by Woods (1970). He went to some length to show that much of the phenomena accounted for by transformational grammar (Chomsky, 1965) could be parsed in ATNs using register testing and setting together with a hold list mechanism. The ATN section of this chapter is a simplified and cleaned-up account of that paper, incorporating later work by Kaplan (1973). Alternative presentations of ATNs can also be found in Bates (1978) and Winograd (1983).

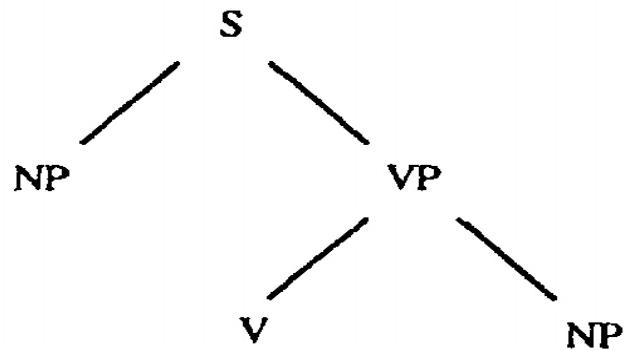
Similar augmentation techniques have been adapted to systems based on CFGs, although in practice many of these systems have accepted many structures that are not reasonable sentences. This is because the grammars are

built so that constituents are optional even in contexts where there could be no movement. The parser then depends on some ad hoc feature manipulation to eliminate some of these false positives, or on semantic interpretation to reject the ill-formed sentences that were accepted. A good example of how much can be done with this approach is provided by Robinson (1982).

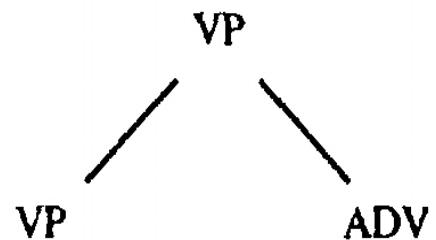
[Allen 1995: Chapter 5 - Grammars for Natural Language 153]

BOX 5.5 Tree-Adjoining Grammar

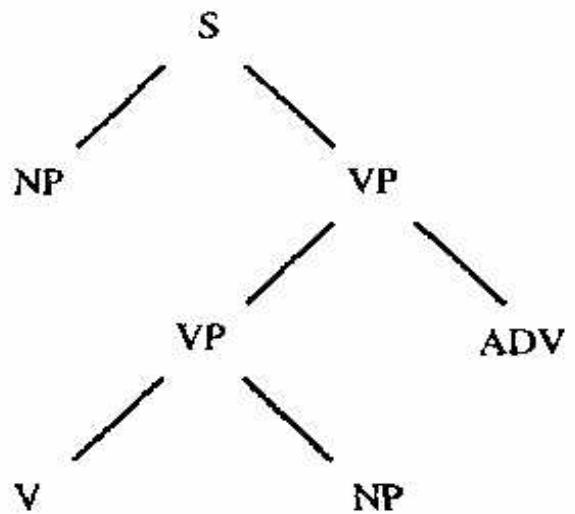
Another approach to handling unbounded dependencies is tree-adjoining grammars (TAGs) (Joshi, 1985). There are no grammar rules in this formalism. Rather, there is a set of initial tree structures that describe the simplest sentences of the language, and a tree operation, called adjoining, that inserts one tree into another to create a more complex structure. For example, a simple initial tree is



More complex sentences are derived using auxiliary trees, which capture the minimal forms of recursion in the language. For example, an auxiliary tree for allowing an adverbial in a verb phrase could be



The adjunction operation involves inserting an auxiliary tree-capturing recursion of some constituent C into another tree that contains a constituent C. Adjoining the auxiliary tree for VP above into the initial tree for S produces the new tree



By basing the formalism on trees, enough context is provided to capture longdistance dependencies. In this theory there is no movement. Instead the constituents start off being close to each other, and then additional structure is inserted between them by the adjoining operation. The result is a formalism of slightly greater power than CFGs but definitely weaker than context-sensitive grammars.

The work on handling movement using gap threading and the definition of extraposition grammars can be found in Pereira (1981). Similar techniques can be found in many current parsers (such as Alshawi, 1992).

[Allen 1995: Chapter 5 - Grammars for Natural Language 154]

The section on using GAP feature propagation to handle movement is motivated from GPSG (Gazdar et al., 1985). In GPSG the propagation of features is determined by a set of general principles (see Box 5.4). Head-driven phrase structure grammar (Pollard and Sag, 1987; 1993) is a descendant of GPSG that is more oriented toward computational issues. It uses subcategorization information on the heads of phrases extensively and, by so doing, greatly simplifies the context-free grammar at the expense of a more complex lexicon.

>> [back](#)

Exercises for Chapter 5

1. (*easy*) Distinguish between bounded (local) movement and unbounded (nonlocal) movement. What extensions were added to the augmentation system to enable it to handle unbounded movement? Why is wh-movement called unbounded movement? Give examples to support your àkint.

2. (*easy*) Expand the following abbreviated constituents and rules into their full feature structure format.

(S [-inv, Q] **AGR** ?a)

NP [R, -gap]

VP → V[_np_s:inf] NP S[inf]

3. (medium) Using the grammar developed in Section 5.3, show the analyses of the following questions in chart form, as shown in Figure 5.12:

In which town were you born?

Where were you born?

When did they leave?

What town were you born in?

4. (medium) GPSG allows certain rules to have multiple head subconstituents. For instance, the rule for a conjunction between two verb phrases would contain two heads:

VP → VP and VP

a. How does the presence of multiple heads affect the algorithm that produces the propagation of the GAP feature? In order to answer this question, consider some examples, such as the following sentences:

Who did you see and give the book to?

What man did Mary hate and Sue love?

Also consider that the following sentences are ill-formed:

*** Who did you see and give the book to John?**

*** What man did Mary hate John and Sue love?**

b. Write out the VP rule showing the GAP features, and then draw the chart for the sentence

[Allen 1995: Chapter 5 - Grammars for Natural Language 155]

Who did Mary see and Sue see?

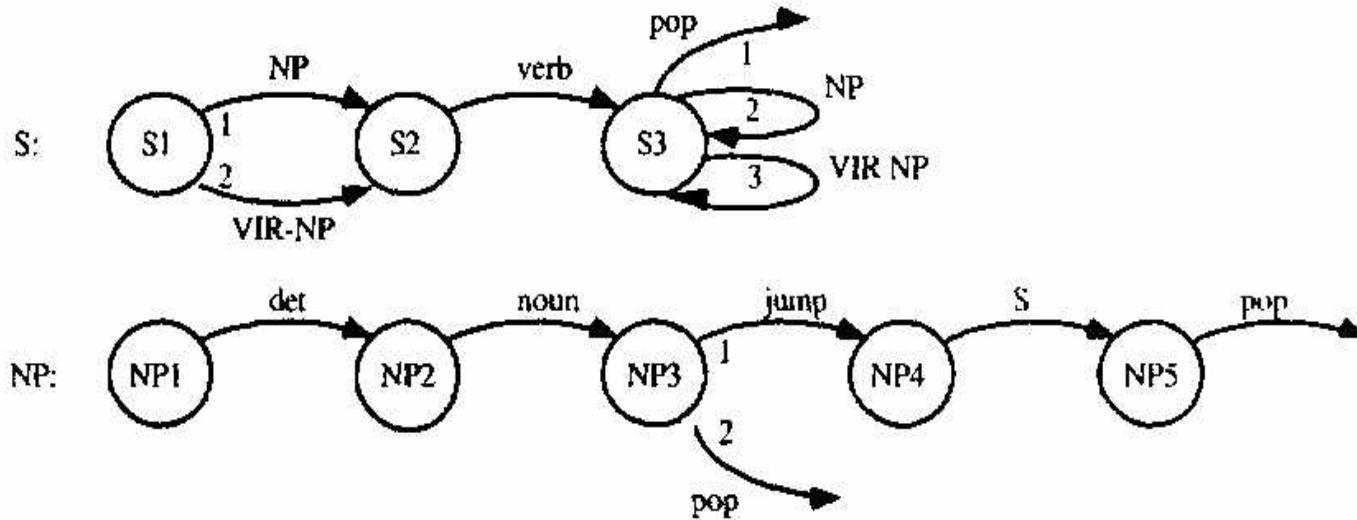
using Grammar 5.8 augmented with your rule. You need only show the constituents that are used in the final analysis, but be sure to show all the feature values for each constituent.

5. (*medium*) Another phenomenon that requires a movement analysis is topicalization, which allows sentences of the form

John, Mary saw.

To John, Mary told the story.

- a. Argue that the same type of constraints that apply to wh-questions apply to topicalization. This establishes that the GAP feature propagation technique can be used.
 - b. Do you need to introduce a new GAP-like feature to handle topicalization, or can the same GAP feature used for questions and relative clauses be re-used?
 - c. Extend Grammar 5.8 to accept the forms of topicalization shown in the examples.
6. (*medium*) Consider the following ATN grammar for sentences. On arc NP3/l there is the action of holding the current noun phrase.



- Give a sequence of legal sentences that can be made arbitrarily long and still be accepted by this ATN.
- Which of the following sentences would be accepted by this ATN (assuming appropriate word categories for words)? For those that are accepted, outline the structure of the sentence as parsed by the net-work (that is, a plausible register value structure).

- i. The man Mary hit hit the ball.
- ii. The man the man hit the man hit.
- iii. The man hit hit.
- iv. Hit the man hit the man.

[Allen 1995: Chapter 5 - Grammars for Natural Language 156]

7. (hard) The following *noun* phrases arise in some dialogues between a computer operator and various users of the computer system.

Could you mount **a magtape** for me?

No ring please.

I am not exactly sure of **the reason**, but we were given **a list of users we are not supposed to mount magtapes for**, and you are on it.

Could you possibly retrieve **the following two files**? I think **they were** on **our directory** last night.

Any chance I can recover from **the most recent system dump**?

Extend the grammar developed in Sections 5.3 and 5.4 so that it accepts these noun phrases. Give lexicon entries for all the new words' in the noun phrases, and show the final charts that result from parsing each NP.

8. (hard)

- a. The grammar for parsing the auxiliary verb structure in English developed in Section 5.1 cannot handle phrases of the following form:

Jack **has to see** a doctor.

The cat **had to be found**.

Joe **has to be winning** the race.

The book **would have had to have been found** by Jack.

Extend the grammar such that it accepts the preceding auxiliary sequences yet does not accept unacceptable sentences such as

- * Jack **has to have to see** a doctor.
- * Janet **had to played** the violin.
- * Will **would to go** to the movies.

b. Perform a similar analysis, showing examples and counterexamples, of the use of phrases of the form **be going to** within the auxiliary system.

>> [back](#)

Allen 1995 : Natural Language Understanding

	<u>Contents</u>	<u>Preface</u>	<u>Introduction</u>	
<u>previous chapter</u>	<u>Part I Syntactic Processing</u>	<u>Part II Semantic Interpretation</u>	<u>Part III - Context / World Knowledge</u>	<u>next chapter</u>
	<u>Appendices</u>	<u>Bibliography</u>		
	<u>Summaries</u>	<u>Further Readings</u>	<u>Exercises</u>	

Chapter 6: Toward Efficient Parsing

	<u>6.1 Human Preferences in Parsing</u>
	<u>6.2 Encoding Uncertainty: Shift-Reduce Parsers</u>
	<u>6.3 A Deterministic Parser</u>
	<u>6.4 Techniques for Efficient Encoding of Ambiguity</u>
	<u>6.5 Partial Parsing</u>
	<u>Summary</u>
	<u>Related work and Further Readings</u>
	<u>Exercises for Chapter 6</u>

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 159]

All the parsing frameworks discussed so far have depended on complete search techniques to find possible interpretations of a sentence. Such search processes do not correspond well with many people's intuitions about human processing of language. In particular, human parsing seems closer to a deterministic process - that is, a process that doesn't extensively search through alternatives but rather uses the information it has at the time to choose the correct interpretation. While intuitions about our own perceptual processes are known to be highly unreliable, experimental evidence also suggests that people do not perform a complete search of a grammar while parsing. This chapter examines some ways of making parsers more efficient, and in some cases, deterministic. If syntactic parsing is not your focus, this entire chapter can be treated as optional material.

There are two different issues that are of concern. The first involves improving the efficiency of parsing algorithms by reducing the search but not changing the final outcome, and the second involves techniques for choosing between different interpretations that the parser might be able to find. Both will be examined in this chapter.

Section 6.1 explores the notion of using preferences in parsers to select certain interpretations, sometimes at the cost of eliminating other possible interpretations. Evidence for this type of model is drawn from psycholinguistic theories concerning the human parsing process. This provides the background for the techniques discussed in the rest of the chapter. Section 6.2 explores one method for improving the efficiency of parsers by pre-compiling much of the search into tables. These techniques are used in shift-reduce parsers, which are used for parsing programming languages. These techniques can be generalized to handle ambiguity and used for natural languages as well. Section 6.3 describes a fully deterministic parsing framework that parses sentences without a recourse to backtracking. This means that the parser might make mistakes and fail to find interpretations for some sentences. If the technique works correctly, however, only sentences that people tend to misparse will be misparsed by the system. Section 6.4 looks at a

variety of techniques to efficiently represent ambiguity by "collapsing" interpretations, or by eliminating ambiguity by redefining the goals of syntactic parsing. Section 6.5 looks at techniques of partially parsing the input by analyzing only those aspects of a sentence that can be determined reliably.

>> [back](#)

6.1 Human Preferences in Parsing

So far this book has discussed parsing in the abstract, without regard to any psychological evidence concerning how people parse sentences. Psycholinguists have conducted many investigations into this issue using a variety of techniques, from intuitions about preferred interpretations to detailed experiments monitoring moment-by-moment processing as subjects read and listen to language. These studies have revealed some general principles concerning how people resolve ambiguity.

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 160]

1.1	$S \rightarrow NP VP$	1.4	$NP \rightarrow ART N$
1.2	$VP \rightarrow V NP PP$	1.5	$NP \rightarrow NP PP$
1.3	$VP \rightarrow V NP$	1.6	$PP \rightarrow P NP$

Grammar 6.1 A simple CFG

Grammar 6.1 A simple CFG

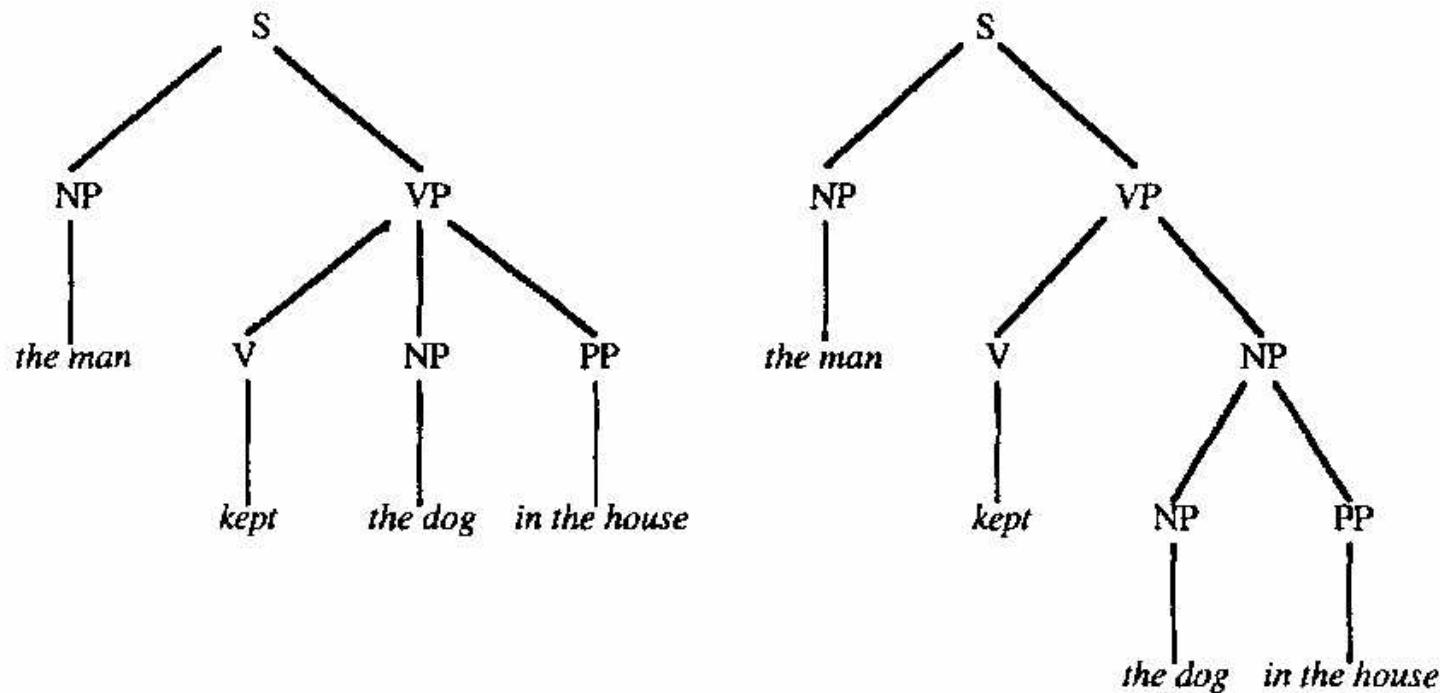


Figure 6.2 The interpretation on the left is preferred by the minimal attachment principle

Figure 6.2 The interpretation on the left is preferred by the minimal attachment principle

The most basic result from these studies is that people do not give equal weight to all possible syntactic interpretations. This can be illustrated by sentences that are temporarily ambiguous, which cause a conscious feeling of having pursued the wrong analysis, as in the sentence "*The raft floated down the river sank*". When you read the word "sank" you realize that the interpretation you have constructed so far for the sentence is not correct. In the literature, such sentences are often called "**garden-path**" sentences, based on the expression about leading someone down a garden path. Here are a few of the general principles that appear to predict when garden paths will arise.

>> [back](#)

Minimal Attachment

The most general principle is called the **minimal attachment** principle, which states that there is a preference for the syntactic analysis that creates the least number of nodes in the parse tree. Thus, given Grammar 6.1, the sentence "*The man kept the dog in the house*" would be interpreted with the PP "*in the house*" modifying the verb rather than the NP "*the dog*". These two interpretations are shown in Figure 6.2. The interpretation with the PP attached to the VP is derived using rules 1.1, 1.2, and 1.6 and three applications of rule 1.4 for the NPs. The parse tree has a total of 14 nodes. The interpretation with the PP attached to the

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 161]

NP is derived using rules 1.1, 1.3, 1.5, and 1.6 and three applications of rule 1.4, producing a total of 15 nodes in the parse tree. Thus this principle predicts that the first interpretation is preferred, which probably agrees with your intuition.

This principle appears to be so strong that it can cause certain sentences to be almost impossible to parse correctly. One example is the sentence

We painted all the walls with cracks.

which, against all common sense, is often read as meaning that cracks were painted onto the walls, or that cracks were somehow used as an instrument to paint the walls. Both these anomalous readings arise from the PP being attached to the VP (*paint*) rather than the NP (*the walls*). Another classic example is the sentence

The horse raced past the barn fell.

which has a reasonable interpretation corresponding to the meaning of the sentence "*The horse that was raced past the barn fell*". In the initial sentence, however, creating a reduced relative clause when the word "*raced*" is encountered introduces many more nodes than the simple analysis where "*raced*" is the main verb of the sentence. Of course, this second interpretation renders the sentence unanalyzable when the word *fell* is encountered.

>> [back](#)

Right Association

The second principle is called right association or late closure. This principle states that, all other things being equal, new constituents tend to be interpreted as being part of the current constituent under construction (rather than part of some constituent higher in the parse tree). Thus, given the sentence

George said that Henry left in his car.

the preferred interpretation is that Henry left in the car rather than that George spoke in the car. Both interpretations are, of course, syntactically acceptable analyses. The two interpretations are shown in Figure 6.3. The former attaches the PP to the VP immediately preceding it, whereas the latter attaches the PP to the VP higher in the tree. Thus the right association principle prefers the former. Similarly, the preferred interpretation for the sentence "*I thought it would rain yesterday*" is that yesterday was when it was thought to rain, rather than the time of the thinking.

>> [back](#)

Lexical Preferences

In certain cases the two preceding principles seem to conflict with each other. In the sentence "*The man kept the dog in the house*", the principle of right association appears to favor the interpretation in which the PP modifies the dog, while the minimal attachment principle appears to favor the PP modifying the VP. You

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 162]

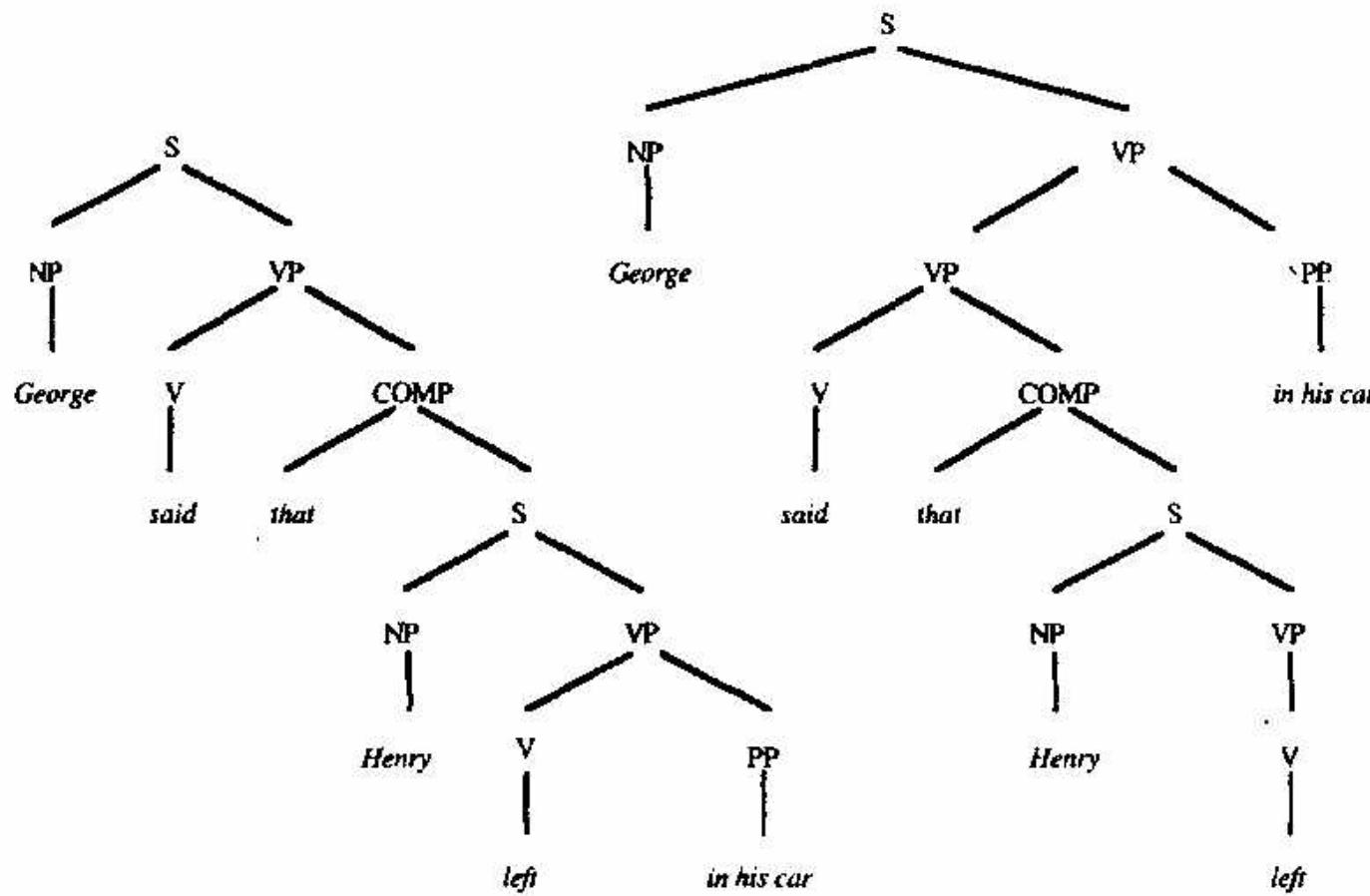


Figure 6.3 Two interpretations of *George said that Henry left in his car.*

Figure 6.3 Two interpretations of "George said that Henry left in his car"

might suggest that minimal attachment takes priority over right association in such cases; however, the relationship appears to be more complex than that. Consider the sentences

1. I wanted the dog in the house.
2. I kept the dog in the house.
3. I put the dog in the house.

The PP "*in the house*" in sentence 1 seems most likely to be modifying "*dog*" (although the other interpretation is possible, as in the sense "*I wanted the dog to be in the house*"). In sentence 2, the PP seems most likely to be modifying the VP (although modifying the NP is possible, as in "*I kept the dog that was in the house*"). Finally, in sentence 3, the PP is definitely attached to the VP, and no alternative reading is possible.

These examples demonstrate that lexical items, in this case the verb used, can influence parsing preferences. In many cases, the lexical preferences will override the preferences based on the general principles. For example, if a verb subcategorizes for a prepositional phrase, then some PP must be attached to the VP. Other PPs might also be identified as having a strong preference for attachment within the VP. If neither of these cases holds, the PP will be attached according to the general principles.

Thus, for the preceding verbs, "*want*" has no preference for any PPs, whereas "*keep*" might prefer PPs with prepositions "*in*", "*on*", or "*by*" to be attached to the VP. Finally, the verb "*put*" requires (subcategorizes for) a PP beginning with "*in*", "*on*", "*by*", and so on, which must be attached to the VP. This approach has promise but is

2.1 $S \rightarrow NP\ VP$
2.2 $NP \rightarrow ART\ N$

2.3 $VP \rightarrow AUX\ V\ NP$
2.4 $VP \rightarrow V\ NP$

Grammar 6.4 A simple grammar with an AUX/V ambiguity

Grammar 6.4 A simple grammar with an AUX/V ambiguity

hard to evaluate without developing some formal framework that allows such information to be expressed in a uniform way. Such techniques will be addressed in the next chapter.

6.2 Encoding Uncertainty: Shift-Reduce Parsers

One way to improve the efficiency of parsers is to use techniques that encode uncertainty, so that the parser need not make an arbitrary choice and later backtrack. Rather, the uncertainty is passed forward through the parse to the point where the input eliminates all but one of the possibilities. If you did this explicitly at parse time, you would have an algorithm similar to the breadth-first parser described in Chapter 3. The efficiency of the technique described in this section arises from the fact that all the possibilities are considered in advance, and the information is stored in a table that controls the parser, resulting in parsing algorithms that can be much faster than described thus far.

These techniques were developed for use with unambiguous context-free grammars - grammars for which there is at most one interpretation for any given sentence. While this constraint is reasonable for programming languages, it is clear that there is no unambiguous grammar for natural language. But these techniques can be extended in various ways to make them applicable to natural language parsing.

Specifying the Parser State

Consider using this approach on the small grammar in Grammar 6.4. The technique involves predetermining all possible parser states and determining the transitions from one state to another. A parser state is defined as the complete set of dotted rules (that is, the labels on the active arcs in a chart

parser) applicable at that position in the parse. It is complete in the sense that if a state contains a rule of the form $\text{Y} \rightarrow \dots \circ \text{x} \dots$, where x is a nonterminal, then all rules for x are also contained in the state. For instance, the initial state of the parser would include the rule

S → o NP VP

as well as all the rules for NP, which in Grammar 6.4 is only

NP → o ART N

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 164]

Thus the initial state, S₀, could be summarized as follows:

Initial State S₀: S → o NP VP

NP → o ART N

In other words, the parser starts in a state where it is looking for an NP to start building an S and looking for an ART to build the NP. What states could follow this initial state? To calculate this, consider advancing the dot over a terminal or a nonterminal and deriving a new state. If you pick the symbol ART, the resulting state is

State S₁: NP → ART o N

If you pick the symbol NP, the rule is

S → NP o VP

in the new state. Now if you expand out the VP to find all its possible starting symbols, you get the following:

State S2: S → NP o VP

VP → o AUX V NP

VP → o V NP

Now, expanding S1, if you have the input N, you get a state consisting of a completed rule:

State S1': NP → ART N o

Expanding 52, a V would result in the state

State S3: VP → V o NP

NP → o ART N

An AUX from S2 would result in the state

State S4: VP → AUX o V NP

and a VP from 52 would result in the state

State S2': S → NP VP o

Continuing from state 53 with an ART, you find yourself in state Si again, as you would also if you expand from S0 with an ART. Continuing from S3 with an NP, on the other hand, yields the new state

State S3': VP → V NP o

Continuing from S4 with a V yields

State S5: VP -> AUX V o NP

NP -> o ART N

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 165]

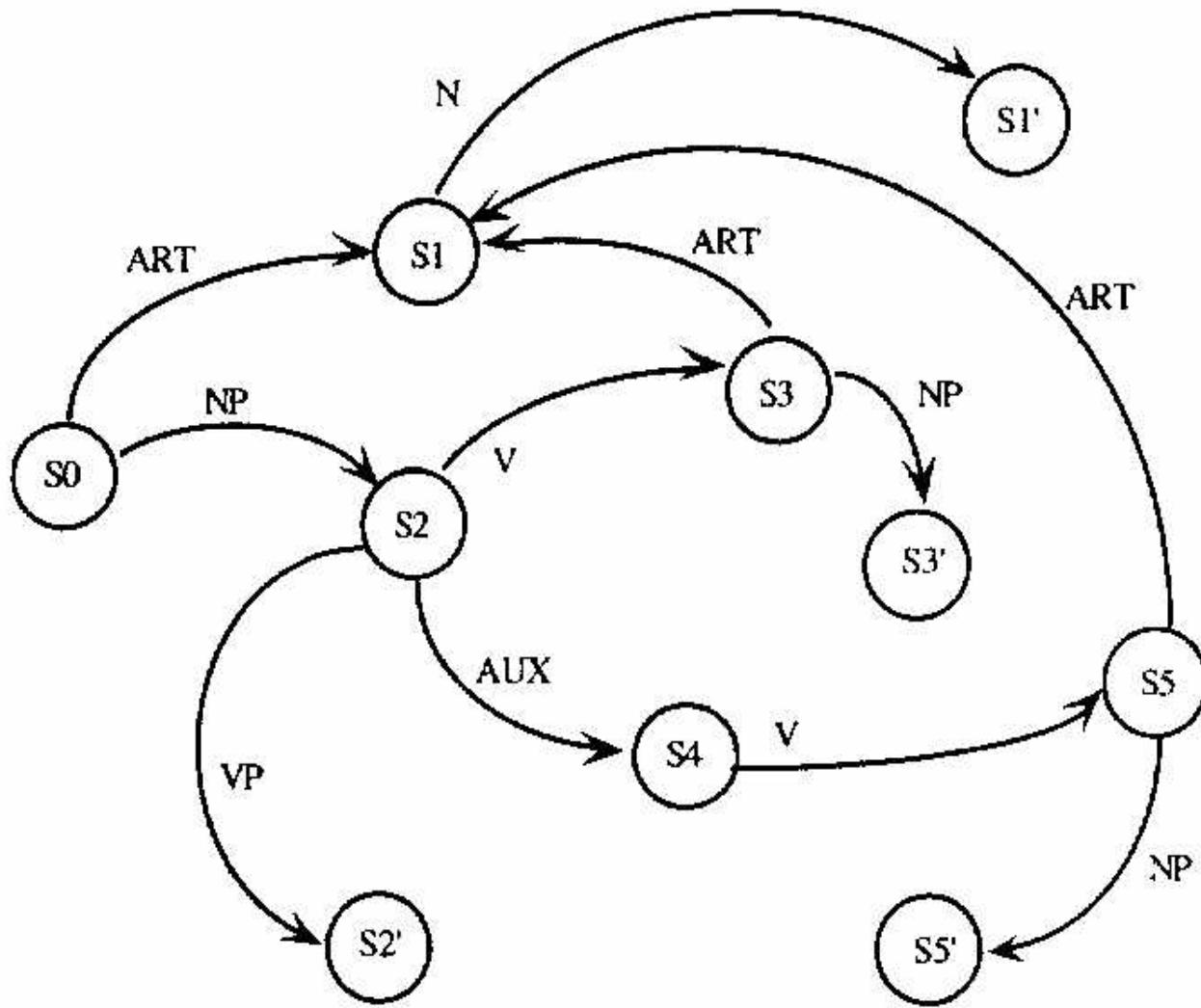


Figure 6.5 A transition graph derived from Grammar 6.1

Figure 6.5 A transition graph derived from Grammar 6.1

and continuing from S5 with an ART would produce state S1 again. Finally, continuing from S5 with an NP would produce the state

```
State S5': VP -> AUX V NP o
```

Now that this process is completed, you can derive a transition graph that can be used to control the parsing of sentences, as is shown in Figure 6.5.

>> [back](#)

A Shift-Reduce Parser

These states can be used to control a parser that maintains two stacks: the **parse stack**, which contains parse states (that is, the nodes in Figure 6.5) and grammar symbols; and the input stack, which contains the input and some grammar symbols. At any time the parser operates using the information

specified for the top state on the parse stack.

The states are interpreted as follows. The states that consist of a single rule with the dot at the far right-hand side, such as S2',

S → NP VP o

indicate that the parser should rewrite the top symbols on the parse stack according to this rule. This is called a reduce action. The newly derived symbol (S in this case) is pushed onto the top of the input stack.

Any other state not containing any completed rules is interpreted by the transition diagram. If the top input symbol matches an arc, then it and the new

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 166]

State	Top Input Symbol	Action	GoTo
S0	ART	Shift	S1
S0	NP	Shift	S2
S0	S	Shift	S0'
S0'	ϵ	Succeed	---
S1	N	Shift	S1'
S1'	-----	Reduce by rule 2.2	---
S2	V	Shift	S3
S2	AUX	Shift	S4
S2	VP	Shift	S2'
S2'	-----	Reduce by rule 2.1	---
S3	ART	Shift	S1
S3	NP	Shift	S3'
S3'	-----	Reduce by rule 2.4	---
S4	V	Shift	S5
S5	ART	Shift	S1
S5	NP	Shift	S5'
S5'	-----	Reduce by rule 2.3	---

Figure 6.6 The oracle for Grammar 6.4

Figure 6.6 The oracle for Grammar 6.4

state (at the end of the arc) are pushed onto the parse stack. This is called the **shift action**. Using this interpretation of states you can construct a table, called the oracle, that tells the parser what to do in every situation. The oracle for Grammar 6.4 is shown in Figure 6.6. For each state and possible input, it specifies the action and the next state. Reduce actions can be applied regardless of the next input, and the accept action only is possible when the input stack is empty (that is, the next symbol is the empty symbol e). The parsing algorithm for using an oracle is specified in Figure 6.7.

Consider parsing "*The man ate the carrot*". The initial state of the parser is

Parse Stack	Input Stack
(S0)	(The man ate the carrot)

Looking up the entry in the table in Figure 6.6 for state S0 for the input ART (the category of the word *the*), you see a shift action and a move to state S1:

Parse Stack	Input Stack
(S1 ART S0)	(man ate the carrot)

Looking up the entry for state S1 for the input N, you see a shift action and a move to state S1:

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 167]

This algorithm uses the following information:

- $Action(S, W)$ —a function that maps a state and an input constituent to one of the values *shift*, *reduce i*, or *accept*
- $GoTo(S, W)$ —a function that maps a state and an input constituent to a new state
- a parse stack of form $(S_n C_n \dots S_1 C_1 S_0)$, where S_i are parse states and C_i are constituents
- an input stack of form $(W_1 \dots W_n)$, where W_i is a constituent symbol or word

The parser operates by continually executing the following steps until success or failure:

1. If $Action(S_n, W_1) = Shift$, and $GoTo(S_n, W_1) = S$, then remove W_1 from the input stack and push it on the parse stack, and then push S onto the parse stack, resulting in the following stacks:

parse stack: $(S W_1 S_n C_n \dots S_1 C_1 S_0)$
input stack: $(W_2 \dots W_n)$

2. If $Action(S_n, W_1) = Reduce i$ and grammar rule i has n constituents on its right-hand side, then remove $2n$ elements from the parse stack, and push the left-hand side of rule i onto the input stack. For example, if rule i were $NP \rightarrow ART\ N$, then the new state would be

parse stack: $(S_{n-2} C_{n-2} \dots S_1 C_1 S_0)$
input stack: $(NP\ W_1 \dots W_n)$

3. If $Action(S_n, W_1) = Accept$, then the parser has succeeded.
4. If $Action(S_n, W_1)$ is not defined, then the parser has failed.

Figure 6.7 The parsing algorithm for a shift-reduce parser

Figure 6.7 The parsing algorithm for a shift-reduce parser

Parse Stack	Input Stack
(S1' N S1 ART S0)	(ate the carrot)

Looking up the entry for state S_i' , you then reduce by rule 2.2, which removes the S_i , N , S_1 , and ART from the parse stack and adds NP to the input stack:

Parse Stack	Input Stack
(S0)	(NP ate the carrot)

Again, consulting the table for state S_0 with input NP, you now do a shift and move to state S_2 :

Parse Stack	Input Stack
(S2 NP S0)	(ate the carrot)

Next, the three remaining words all cause shifts and a move to a new state, ending up with the parse state:

Parse Stack	Input Stack
(S1' N S1 ART S3 V S2 NP S0)	()

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 168]

The reduce action by rule 2.2 specified in state S1' pops the N and ART from the stack (thereby popping S1 and S1' as well), producing the state:

Parse Stack	Input Stack
(S3 V S2 NP S0)	(NP)

You are now back at state S3, with an NP in the input, and after a shift to state S3', you reduce by rule 2.4, producing:

Parse Stack	Input Stack

Parse Stack

Input Stack

(S2 NP S0)

(VP)

Finally, from state S2 you shift to state S2' and reduce by rule 2.1, producing:

Parse Stack

Input Stack

(S0)

(S)

From this state you shift to state S0' and are in a position to accept the sentence.

You have parsed the sentence without ever trying a rule in the grammar incorrectly and without ever constructing a constituent that is not part of the final analysis.

>> [back](#)

Shift-Reduce Parsers and Ambiguity

Shift-reduce parsers are efficient because the algorithm can delay decisions about which rule to apply. For example, if you had the following rules for parsing an

1. **NP → ART N REL-PRO VP**

2. **NP → ART N PP**

a top-down parser would have to generate two new active arcs, both of which would be extended if an ART were found in the input. The shift-reduce parser, on the other hand, represents both using a single state, namely

- NP1:** **NP → o ART N REL-PRO VP**
-
- NP → o ART N PP**

If an ART is found, the next state will be

- NP2:** **NP → ART o ART N REL-PRO VP**
-
- NP → ART o N PP**

Thus the ART can be recognized and shifted onto the parse stack without committing to which rule it is used in. Similarly, if an N is seen at state NP2, the state is

NP3: **NP → ART N o REL-PRO VP**

NP → ART N o N PP

PP → o P NP

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 169]

Now from NP3, you finally can distinguish the cases. If a REL-PRO is found next, the state is

NP4: **NP → ART N REL-PRO o VP**

VP → o V NP

which leads eventually to a reduction by rule 1. If a P is found, you move to a different state, which eventually builds a PP that is used in the reduction by rule 2. Thus the parser delays the decision until sufficient information is available to choose between rules.

>> [back](#)

Lexical Ambiguity

This ability to postpone decisions can be used to deal with some lexical ambiguity by a simple extension to the parsing process. Whereas earlier you classified a lexical entry when it was shifted (that is, "*carrot*" was converted to N during the shift), you now allow ambiguous words to be shifted onto the parse stack as they are, and delay their categorization until a reduction involving them is made.

To accomplish this extension, you must expand the number of states to include states that deal with ambiguities. For instance, if "*can*" could be a V or an AUX, the oracle cannot determine a unique action to perform from state 52—if it were a V you would shift to state S3 and if it were an AUX you would shift to state S4. Such ambiguities can be encoded, however, by generating a new state from S2 to cover both possibilities simultaneously. This new state will be the union of states S3 and S4:

S3-4 : VP → ART o AUX V NP

VP → V o NP

NP → o ART N

In this case the next input should resolve the ambiguity. If you see a V next, you will move to S5 (just as you would from state 54). If you see an ART next, you will move to S1, and if you see an NP next, you will move to S3' (just as you would from S3). Thus the new state maintains the ambiguity long enough for succeeding words to resolve the problem. Of course, in general, the next word might also be ambiguous, so the number of new states could be quite large. But if the grammar is unambiguous - that is, there is only one possible interpretation of the sentence once it is completely parsed - this technique can be used to delay the decisions as long as necessary. (In fact, for those who know automata theory, this process is simply the construction of a deterministic simulation of a nondeterministic finite automata.)

>> [back](#)

Ambiguous Parse States

There are, however, other forms of ambiguity that this parser, as it now stands, is not able to handle. The simplest examples involve the ambiguity that arises when one rule is a proper prefix of another, such as the rules

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 170]

3. **NP → ART N**

4. **NP → ART N PP**

With these two rules, we will generate a parse state containing the two dotted rules

NP → ART o N

NP → ART o N PP

If the next input in this state is an N, you would move to a state consisting of

NP5: NP → ART o

NP → ART N o PP

PP → o P NP

But now there is a problem. If the next input is a P, eventually a PP will be built and you return to state NP5. But the parser cannot decide whether to reduce by rule 3, leaving the PP to be attached later, or to shift the PP (and then reduce by rule 4). In any reasonable grammar of English, of course, both choices might lead to acceptable parses of the sentence. There are two ways to deal with this problem. The first strategy maintains a deterministic parser

and accepts the fact that it may misparse certain sentences. The goal here is to model human parsing performance and fail on only those that would give people trouble. One recent suggestion claims the following heuristics favor more intuitive interpretations over less intuitive ones:

- favor shift operations over reduce operations
- resolve all reduce-reduce conflicts in favor of the longest rule (that is, the one that uses the most symbols from the stack)

Using these strategies, you could build a deterministic parser that picks the most favored rule and ignores the others. It has been claimed that the first heuristic corresponds to the right-association preference and the second to the minimal -attachment preference. While convincing examples can be shown to illustrate this, remember that the strategies themselves, and thus the examples, are highly sensitive to the form of the grammar used.

The second approach to dealing with the ambiguous states is to abandon the deterministic requirement and reintroduce a search. In this case you could consider each alternative by either a depth-first search with backtracking or a breadth-first search maintaining several interpretations in parallel. Both of these approaches can yield efficient parsers. The depth-first approach can be integrated with the preference strategies so that the preferred interpretations are tried first.

>> [back](#)

6.3 A Deterministic Parser

A deterministic parser can be built that depends entirely on matching parse states to direct its operation. Instead of allowing only shift and reduce actions, however, a richer set of actions is allowed that operates on an input stack called the buffer.

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 171]

The Parse Stack

Top → (S SUBJ (NP DET the
HEAD cat))

The Buffer

<i>ate</i>	<i>the</i>	<i>fish</i>
------------	------------	-------------

Figure 6.8 A situation during a parse

Figure 6.8 A situation during a parse

Rather than shifting constituents onto the parse stack to be later consumed by a reduce action, the parser builds constituents incrementally by attaching buffer elements into their parent constituent, an operation similar to feature assignment. Rather than shifting an NP onto the stack to be used later in a reduction $S \rightarrow NP\ VP$, an S constituent is created on the parse stack and the NP is attached to it. Specifically, this parser has the following operations:

- Create a new node on the parse stack (to push the symbol onto the stack)
- Attach an input constituent to the top node on the parse stack
- Drop the top node in the parse stack into the buffer

The drop action allows a completed constituent to be reexamined by the parser, which will then assign it a role in a higher constituent still on the parse stack. This technique makes the limited lookahead technique surprisingly powerful.

To get a feeling for these operations, consider the situation in Figure 6.8, which might occur in parsing the sentence "*The cat ate the fish*". Assume that the first NP has been parsed and assigned to the SUBJ feature of the S constituent on the parse stack. The operations introduced earlier can be used to complete the analysis. Note that the actual mechanism for deciding what operations to do has not yet been discussed, but the effect of the operations is shown here to provide intuition about the data structure. The operation

Attach to MAIN-V

would remove the lexical entry for *ate* from the buffer and assign it to the MAIN-V feature in the S on the parse stack. Next the operation

Create NP

would push an empty NP constituent onto the parse stack, creating the situation in Figure 6.9.

Next the two operations

Attach to DET

Attach to HEAD

would successfully build the NP from the lexical entries for "*the*" and "*fish*". The input buffer would now be empty.

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 172]

The Parse Stack

Top → (NP)

(S SUBJ (NP DET the
HEAD cat)
MAIN-V ate)

The Buffer

<i>the</i>	<i>fish</i>	
------------	-------------	--

Figure 6.9 After creating an NP

Figure 6.9 After creating an NP

The Parse Stack

Top → (S SUBJ (NP DET the
HEAD cat)
MAIN-V ate)

The Buffer

(NP DET the HEAD fish)		
---------------------------	--	--

Figure 6.10 After the drop action

Figure 6.10 After the drop action

>> [back](#)

The operation

Drop

pops the NP from the parse stack and pushes it back onto the buffer, creating the situation in Figure 6.10.

The parser is now in a situation to build the final structure with the operation

Attach to OBJ

which takes the NP from the buffer and assigns it to the **OBJ** slot in the S constituent.

Three other operations prove very useful in capturing generalizations in natural languages:

- Switch the nodes in the first two buffer positions
- Insert a specific lexical item into a specified buffer slot
- Insert an empty NP into the first buffer slot

Each rule has a pattern that contains feature checks on the buffer to determine the applicability of the rule. Rules are organized into **packets**, which may be activated or deactivated during the parse. Additional actions are available for changing the parser state by selecting which packets to use. In particular, there are actions to

- **Activate** a packet (that is, all its rules are to be used to interpret the next input)
- **Deactivate** a packet

Pattern	Actions	Priority
Packet BUILD-AUX:		
1. <=AUX, HAVE> <=V, pastprt>	Attach to PERF	10
2. <=AUX, BE> <=V, ing>	Attach to PROG	10
3. <=AUX, BE> <=V, pastprt>	Attach to PASSIVE	10
4. <=AUX, +modal> <=V, inf>	Attach to MODAL	10
5. <=AUX, DO> <=V, inf>	Attach to DO	10
6. <true>	Drop	15

Grammar 6.11 The rules for packet BUILD-AUX

Grammar 6.11 The rules for packet BUILD-AUX

The active packets are associated with the symbols on the parse stack. If a new constituent is created (that is, pushed on the stack), all the active packets associated with the previous top of the stack become inactive until that constituent is again on the top of the stack. Consequently, the drop action will always deactivate the rules associated with the top node. Packets play a role similar to the states in the shift-reduce parser. Unlike the states in the shift-

reduce parser, more than one packet may be active at a time. Thus there would be no need to create packets consisting of the union of packets to deal with word ambiguity, since both can be active simultaneously.

Consider the example rules shown in Grammar 6.11, which deals with parsing the auxiliary structure. The pattern for each rule indicates the feature tests that must succeed on each buffer position for the rule to be applicable. Thus the pattern $<=AUX, HAVE> <=V, pastprt>$ is true only if the first buffer is an AUX structure' with ROOT feature value HAVE, and the second is a V structure with the VFORM feature pastprt. The priority associated with each rule is used to decide between conflicting rules. The lower the number, the higher the priority. In particular, rule 6, with the pattern $<true>$, will always match, but since its priority is low, it will never be used if another one of the rules also matches. It simply covers the case when none of the rules match, and it completes the parsing of the auxiliary and verb structure.

Figure 6.12 shows a parse state in which the state BUILD-AUX is active. It contains an AUXS structure on the top of the stack with packet BULLD-AUX active, and an S structure below with packets PARSE-AUX and CPOOL that will become active once the AUXS constituent is dropped into the buffer.

Given this situation and the rules in Figure 6.11, the parser's next action is determined by seeing which rules match. Rules 1 and 6 succeed, and I is chosen because of its higher priority. Applying the actions of rule 1 produces the state in Figure 6.13. Now the rules in BUILD-AUX are applied again. This time only rule 6 succeeds, so the next action is a drop, creating the state in Figure 6.14.

At this stage the rules in packets PARSE-AUX and CPOOL are active; they compete to determine the next move of the parser (which would be to attach the AUXS structure into the S structure).

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 174]

The Parse Stack

Nodes

Top → (AUXS)

(S MOOD DECL
SUBJ (NP NAME John))

Active Packets

(BUILD-AUX)

(PARSE-AUX CPOOL)

The Input Buffer

(AUX ROOT HAVE FORM pres NUM {3s})	(V ROOT SEE FORM en)	(ART ROOT A NUM {3s})
--	-------------------------	--------------------------

Figure 6.12 A typical state of the parser

Figure 6.12 A typical state of the parser

The Parse Stack

Nodes

Top → (AUXS PERF has)

(S MOOD DECL

(S SUBJ (NP NAME John))

Active Packets

(BUILD-AUX)

(PARSE-AUX CPOOL)

The Input Buffer

(V ROOT SEE VFORM pastprt)	(ART ROOT A NUM {3s})	(N ROOT DAY NUM {3s})
-------------------------------	--------------------------	--------------------------

Figure 6.13 After rule 1 is applied

Figure 6.13 After rule 1 is applied

The Parse Stack

Nodes

Top → (S MOOD DECL
SUBJ (NP NAME John))

Active Packets

(PARSE-AUX CPOOL)

The Input Buffer

(AUXS PERF has)	(V ROOT SEE VFORM pastprt)	(ART ROOT A NUM {3s})
-----------------	-------------------------------	--------------------------

Figure 6.14 The parse state after a drop action

Figure 6.14 The parse state after a drop action

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 175]

The lookahead rules are restricted to examining at most the first three buffer elements, with one exception that occurs when an NP subconstituent needs to be constructed. If the NP starts in the second or third buffer position, then while it is being parsed, that position is used as though it were the first buffer. Called attention shifting, this circumstance is the only exception to the three -buffer restriction. With this qualification a rule may still inspect only three buffer positions, but the starting position may be shifted. Attention shifting is restricted so that under no circumstances can a rule inspect a position beyond the first five.

One of the more interesting things about this parser is the way it captures many linguistic generalizations in an elegant fashion by manipulating the input buffer. For example, consider what is needed to extend a grammar that parses assertions into one that parses yes/no questions as well. In Chapter 5 you handled yes/no questions by adding a few rules to the grammar that handled the initial AUX and NP, and then connecting back into the grammar for assertions. This present parser actually reuses the rules for the initial subject and auxiliary as well. In particular, it uses the switch action to reorder the subject NP and the AUX in the buffer, directly capturing the intuition of subject-aux inversion.

>> [back](#)

Psychological Validity

Because of the limits on the lookahead and the deterministic nature of this parser, you can examine in detail its limitations as well as its coverage. Some researchers have argued that the limitations of the mechanism itself account for various constraints on the form of language, such as the complex-NP constraint described in Chapter 5. Rather than having to impose such a constraint in the grammar, this parser, because of its limitations, could not operate in any other way.

Because of the limited lookahead, this mechanism must commit to certain structural analyses before the entire sentence has been examined. In certain cases a sentence may begin in such a way that the wrong decision is made and the sentence becomes unparsable. These examples were referred to earlier as garden-path sentences. The interesting thing about this phenomenon is that it provides a concrete proposal that can be experimentally validated. In particular, you might ask if the sentence structures with which this parser has difficulty are the same ones with which people have trouble. While the answer is not yet clear, the fact that the question can be asked means that this framework can be investigated experimentally.

In essence, the theory is that any sentence that retains an ambiguity over more than a three-constituent Window may cause trouble with a reader. Note that the lookahead is three constituents, not words; thus an ambiguity might be retained for quite some time without causing difficulty. For example, the following two sentences are identical for the first seven words:

Have the students who missed the exam take it today.

Have the students who missed the exam taken it today?

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 176]

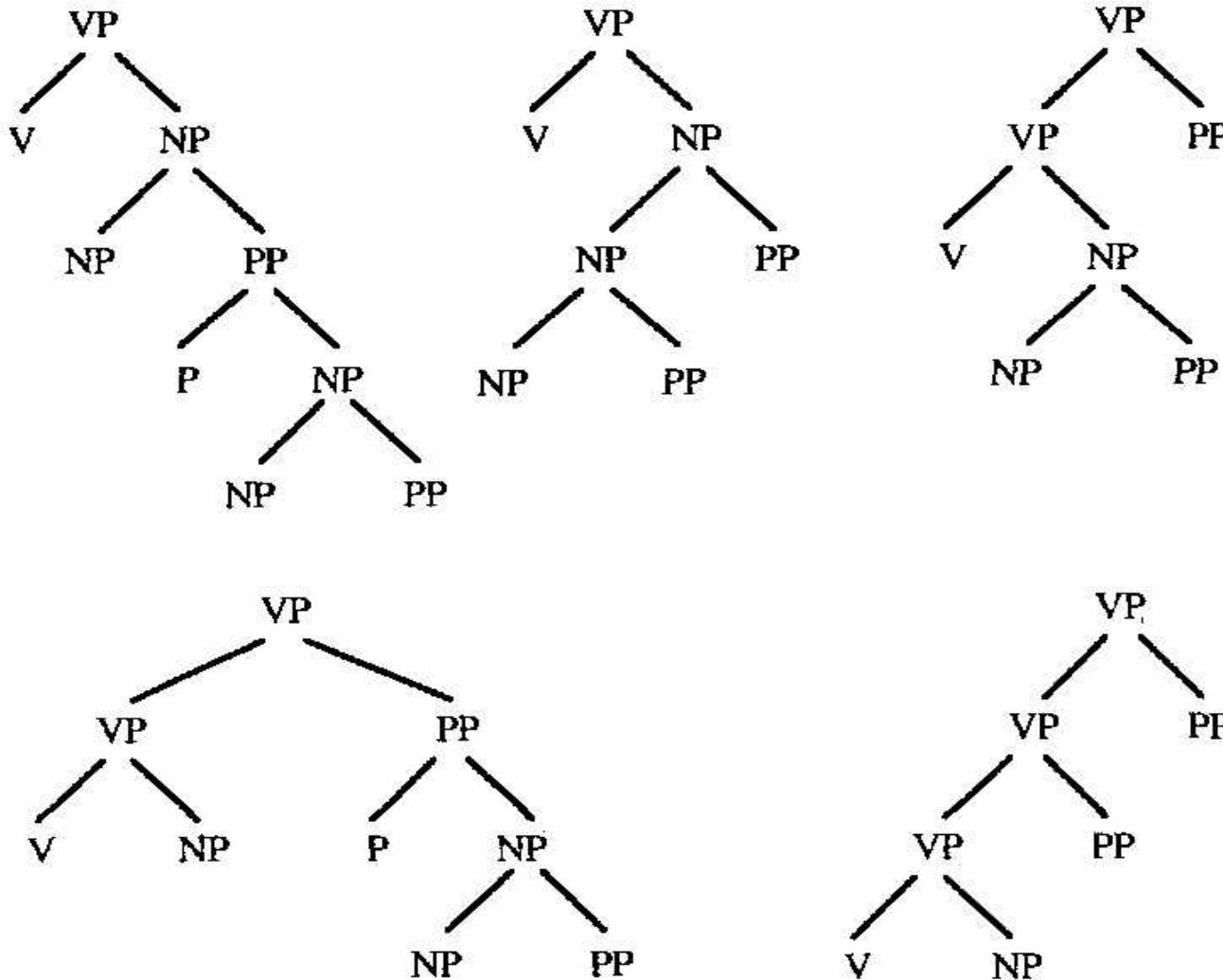


Figure 6.15 Five interpretations of *saw the man in the house with a telescope*

Figure 6.15 Five interpretations of "saw the man in the house with a telescope"

The ambiguity between being an imperative or a sentence versus a yes/no question, however, never extends beyond three constituents, because six of the words are in a single noun phrase. Thus the parser will reach a state where the following two tests can easily distinguish the cases:

```
<=have><=NP><=V, VFORM=base> -> imperative
```

```
<=have><=NP><=V, VFORM=pastprt> -> yes/no question
```

On the other hand, a sentence such as

Have the soldiers given their medals by their sweethearts.

cannot be disambiguated using only three constituents, and this parser, like most people, will initially misinterpret the sentence as a yes/no question that is ill formed, rather than recognize the appropriate imperative reading corresponding to *Have their sweethearts give the soldiers their medals*.

>> [back](#)

6.4 Techniques for Efficient Encoding of Ambiguity

Another way to reduce ambiguity is to change the rules of the game by redefining the desired output. For instance, a significant amount of the ambiguity in

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 177]

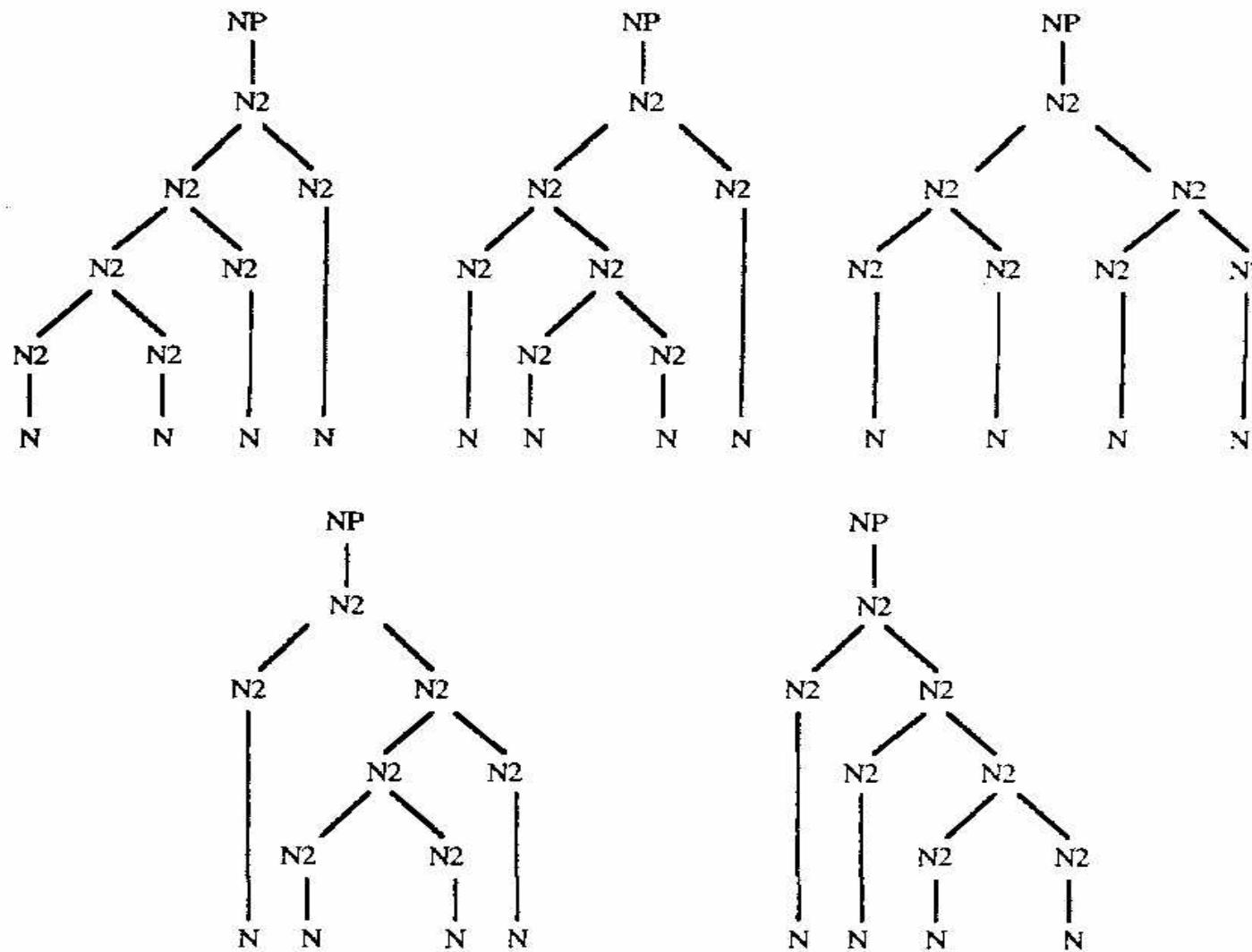


Figure 6.16 Five interpretations of *pot holder adjustment screw*

Figure 6.16 Five interpretations of "pot holder adjustment screw"

sentences results from issues like attachment ambiguity and constructs such as coordination that often have many structural interpretations of essentially the same information. Figure 6.15 shows an example of the alternative interpretations that arise from PP attachment ambiguity in the VP "*saw the man in the house with a telescope*". There are five interpretations arising from the different attachments, each having a different semantic interpretation. Figure 6.16 shows an example of ambiguities in noun-noun modification, such as §*pot holder adjustment screw*§, that would arise from a grammar such as

5. **NP** → **N2**
6. **N2** → **N**
7. **N2** → **N2 N2**

These complications of course interact with each other, so a VP with two prepositional phrases and an NP with a sequence of four nouns would have 25 interpretations, and so on. It is not unusual for a moderate-size sentence, say of 12 words, with a reasonable grammar, to have over 1000 different structural

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 178]

VP		21(1,13)									
VP		20(1,12)									
VP		19(16,8)									
NP		18(15,8)									
VP		17(14,10)									
VP	16(14,7)										
VP	15(1,11)										
VP	14(1,2)										
NP		13(11,8)									
NP		12(2,10)									
NP	11(2,7)										
	PP	10(3,9)									
	NP	9(4,8)									
	PP	7(3,4)	PP	8(5,6)							
V	1	NP	2	P	3	NP	4	P	5	NP	6
saw		the man		in		the house		with		a telescope	

Figure 6.17 The chart for the interpretations of *saw the man in the house with a telescope*

Figure 6.17 The chart for the interpretations "*of saw the man in the house with a telescope*"

interpretations! Clearly some techniques must be introduced to help manage such complications.

This section briefly examines techniques for representing such large quantities of interpretations efficiently. In fact, the chart data structure used so far is already a significant step towards this goal, as it allows constituents to be shared across interpretations. For example, the PP attachment interpretations shown in Figure 6.15 are represented in the chart shown in Figure 6.17. Each chart entry is numbered, and the subconstituents are listed in parentheses. For example, the NP covering "*the man in the house*" is constituent 11 and has subconstituents 2 and 7. While Figure 6.15 uses 32 nonlexical nodes, the chart representation of the same five interpretations uses only 21 nonlexical nodes. While the savings are considerable, and grow as the amount of ambiguity grows, they are often not sufficient to handle the problem.

Another technique that is used is called **packing**. This technique takes advantage of a property that may have occurred to you in looking at Figure 6.17,

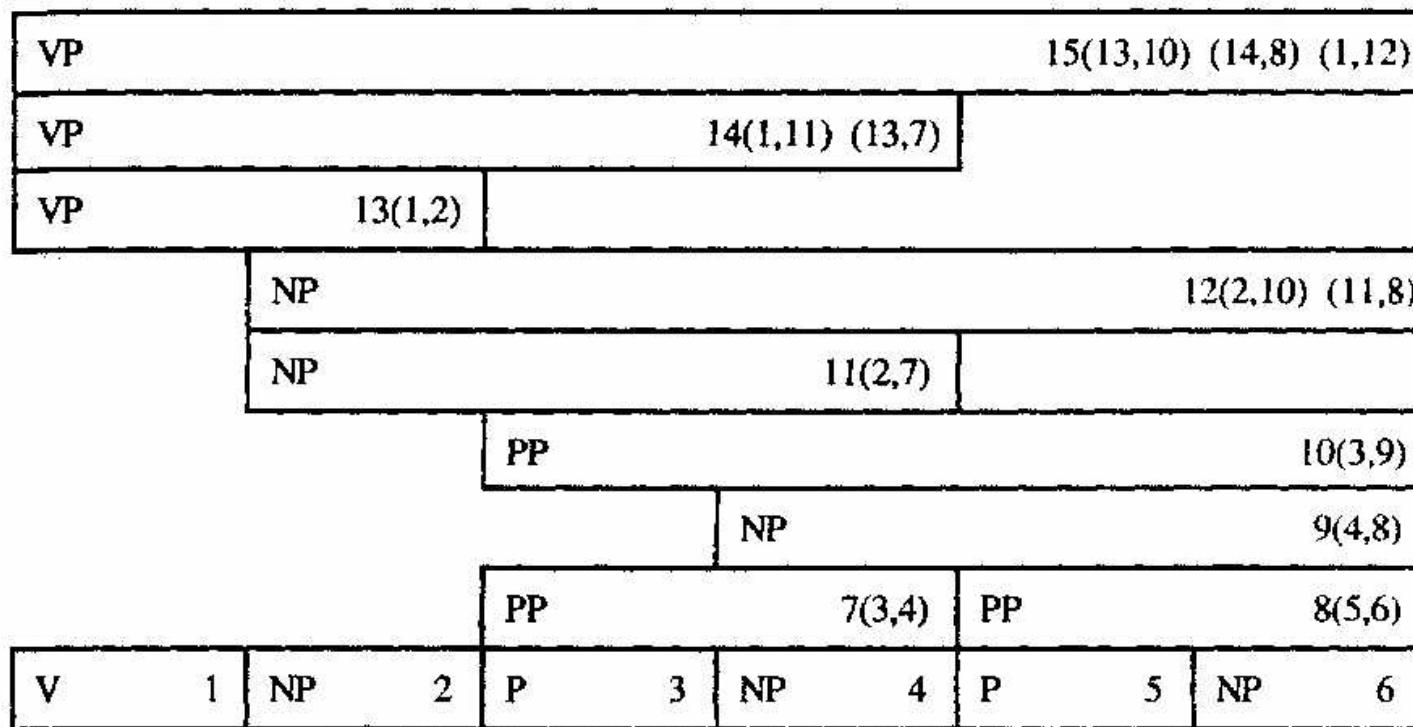


Figure 6.18 A packed chart for the interpretations of *saw the man in the house with a telescope*

Figure 6.18 A packed chart for the interpretations of "saw the man in the house with a telescope"

namely that there are many constituents of the same type that cover exactly the same input. To a parser without features, each of these constituents would be treated identically by any rule that uses them as a subconstituent. A packed chart representation stores only one constituent of any type over the same input, and any others found are collapsed into the existing one. You can still maintain a record for each derivation found in the single constituent. The chart for the same sentence using the packing technique is shown in Figure 6.18 and uses only 15 nonlexical nodes. Packing produces quite efficient representations of interpretations without losing information. With grammars that use feature equations, however, the situation is more complicated. In particular, two VP interpretations covering the same input but having different feature values cannot be merged, because they might differ in how they combine to form larger constituents. You could modify the feature checking to allow success if any one of the possible constituents meets the requirements, but then the chart might overgenerate interpretations, and you would have to reparse the possible solutions at the end to verify that they are valid.

Another way to reduce interpretations is to modify the grammar so that the ambiguities become semantic ambiguities rather than syntactic ambiguities. For example, with noun-noun modification, you might not use rules 5, 6, and 7 mentioned previously, which impose a binary structure onto the noun modifiers, but rather just use a rule that allows a sequence of noun modifiers in a flat structure. In an extended grammar that allows the Kleene + operator, one rule would suffice:

N2 → **N+**

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 180]

The N+ notation allows one or more repetitions of the symbol N. Without the + notation, the grammar would need a series of rules, up to some fixed number of modifiers that you would allow, such as

N2 → **N**

N2 -> N

N2 -> N N

N2 -> N N N

N2 -> N N N N

While such a grammar looks more complicated, it is much more efficient to parse, as the structural ambiguity is eliminated. In particular, there is only one interpretation of the NP "*pot holder adjustment screw*".

Similar approaches have been suggested for attachment ambiguity problems. Rather than generating all interpretations, a single interpretation is constructed, say the one where the PP takes the widest scope. This is sometimes called the **canonical interpretation**. The semantic interpreter then needs to be designed so that it can attach the PP lower in the tree, as seems appropriate based on the semantic constraints.

Another approach is called the **D-theory** or description-theory approach. In D-theory, the meaning of the output of the parser is modified. In the traditional view, used throughout this book, constituents are built by defining their immediate subconstituents. The mother constituent **immediately dominates** its sub-constituents. D-theory defines a transitive relation **dominates** as follows: A constituent M dominates a constituent C if C is an immediate subconstituent of M, of a subconstituent of a subconstituent of M, and so on. With this new terminology, you can say that the output of a traditional parser is a set of immediate dominance relationships that define the parse tree. In D-theory the output is taken as a set of dominance relationships. Thus, for example, the analysis might say that VP1 dominates PP1. This means that PP1 may be a subconstituent of VP1 but also allows that it might be a subconstituent of a subconstituent of VP1. Thus by changing the nature of the output, many forms of ambiguity can be captured concisely.

>> [back](#)

6.5 Partial Parsing

The most radical approach to the ambiguity problem is to give up on producing complete parses of sentences and only look for fragments that can be reliably identified. A simple way to approximate such a parser would be to remove rules from a general grammar that lead to ambiguity problems.. For instance, if you remove the rules

VP → VP PP

NP → NP PP

from a grammar, the PP attachment problem basically disappears. Of course, you also cannot produce a full parse for most sentences. With this simplified gram-

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 181]

VP		7(1,2)		PP		8(3,4)		PP		9(5,6)	
V	1	NP	2	P	3	NP	4	P	5	NP	6

Figure 6.19 Chart for *saw the man in the house with a telescope* showing partial analysis

Figure 6.19 Chart for *saw the man in the house with a telescope* showing partial analysis

mar, a bottom-up chart parser would construct a sequence of syntactic fragments of the sentence that could be useful for later processing. Figure 6.19 shows the final chart for the verb phrase "*saw the man in the park with a telescope*" using a grammar with the PP modifier rules removed. While much of the global structure is missing, all the local structure is unambiguously identified and the chart contains a lot of useful syntactic information.

Of course, this approach in itself doesn't solve the problem; rather, it delays it until the semantic-interpretation phase. But it may be that the ambiguity problem is easier to solve at the semantic level, as there are more sources of information available. Certainly in limited domain systems this appears to be the case. But a detailed discussion of this must wait until semantic interpretation is introduced. These issues will be developed further in Chapter 10. For the moment, consider how much structure can be identified reliably.

From experience in building such systems, we know that certain structures can be identified quite reliably. These include

the noun group - consisting of noun phrases from the initial determiner through prenominal modifiers to the head noun, but not including postmodifiers such as prepositional phrases

the verb group - consisting of the auxiliary verb sequence, some adverbials, up to the head verb

proper noun phrases - including simple names such as "*John*" as well as more complex names like the "*New York Times*". This is especially easy if the input follows standard capitalization conventions because, with capitalization, names can be parsed even if they are not in the lexicon.

It might appear that other structures could be identified reliably as well, such as prepositional phrases, and verb phrases consisting of obligatory sub-categorized constituents. Such phrases can be generated, but they may not be truly part of the full parse of the sentence because of the limitations in handling noun phrases. In particular, a partial parser might suggest that there is a prepositional phrase "*by the leader*" in the sentence "*We were punished by the leader of the group*". But the full parse of the sentence would not contain such a PP. as it would have a PP with the object of the preposition being "*the leader of the group*". Since the partial parser is not able to make attachment decisions, it cannot create the appropriate reading. This leaves you with a choice. Some systems identify the

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 182]

```
(PRO "We")
  (VP  (V "saw")
        (NP "the house boats"))
  (PP  (P "near")
        (NP "the lake"))
  (V "sink")
  (?? "unexpectedly")
  (PP  (P "at")
        (NP "dawn"))
```

Figure 6.20 A partial parse of *We saw the house boats near the lake sink unexpectedly at dawn.*

Figure 6.20 A partial parse of *We saw the house boats near the lake sink unexpectedly at dawn.*

prepositions but leave the PPs unanalyzed, while others construct the incorrect interpretation and depend on the semantic interpretation process to correct it.

Because of the limitations of their coverage, partial parsing systems can be based on regular grammars (or equivalently, finite state machines) rather than using the full power of context-free grammars. They also often depend on a subsystem to accurately predict the correct part of speech for each

word. Such part-of-speech tagging systems are discussed in the next chapter. For the moment, you can simply assume that the correct part of speech is given with the input.

The output of such systems is then a sequence of syntactic fragments, some as small as the lexical category of a function word, and others fairly complex verb sequences and NPs. When faced with an unknown word, the partial parser simply leaves it unanalyzed and moves on. Even with a limited grammar there is potential ambiguity, and partial parsers generally use heuristics to reduce the possibilities. One popular technique is to favor longer constituents over shorter ones of the same type. The heuristic would rather interpret "*The house boats*" as one NP than two (*The house* and *boats*), even though the latter would be possible.

Figure 6.20 shows the sequence of constituents that might be built for such a system given the input "*We saw the house boats near the lake sink unexpectedly at dawn*".

>> [back](#)

Summary

There are several ways to build efficient parsers. Some methods improve the efficiency of search-based parsing models, while others specify models of parsing that are inherently deterministic or only partially analyze the sentences. The methods use several different techniques:

- encoding ambiguity within parse states
- using lookahead techniques to choose appropriate rules
- using efficient encoding techniques for encoding ambiguity
- changing the definition of the output so that it ignores ambiguity

>> [back](#)

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 183]

Related Work and Further Readings

There is a large literature in psycholinguistics on parsing preferences, starting with the work by Kimball (1973). Lexical preferences were suggested by Ford, Bresnan, and Kaplan (1982). An excellent collection of recent work in the field is Clifton, Frazier, and Rayner (in press). Some of the articles in this volume focus on exactly the parsing principles and preferences discussed in Section 6. 1. The researchers seem to be converging on the conclusion that the human processing system takes into account the frequency with which different words and structures appear in different syntactic environments, and then evaluates the most likely possibilities using semantic information and knowledge of the discourse context. In this view, principles such as minimal attachment and right association provide useful descriptions of the preferences, but the preferences actually arise because of other factors. Computational models that have explicitly involved parsing preferences include Shieber (1984), Pereira (1985), and Schubert (1986). Additional computational models will be discussed in Chapter 7.

As mentioned in the text, shift-reduce parsers have traditionally been used for parsing programming languages with unambiguous grammars. A good text discussing this work is Aho et al. (1986), who would classify the shift-reduce parser described in Section 6.2 as an LR(1) parser (that is, a left-to-right parser using a one-symbol lookahead). These techniques have been generalized to handle ambiguity in natural language by various techniques. Shieber (1984) introduces the techniques, described in Section 6.2, that generalize the treatment of terminal symbols to delay lexical classification of words, and that use parsing preferences to select between rules when more than one rule appears to be applicable. Tomita (1986) reincorporates a full search with an optimized parse-tree representation while taking advantage of the speed of the oracle-driven shift-reduce parsing.

The section on deterministic parsing is based on Marcus (1980). Charniak (1983) developed a variant in which the parse states were automatically generated from a context-free grammar. The most comprehensive grammar within this formalism is the Fidditch parser, of which it is hard to find a description in the literature, but see Hindle (1989) for a brief description. Berwick (1985) used this framework in an investigation of language learning.

The technique of packing is used in many systems. Two recent examples are Tomita (1986) and Alshawi (1992). The extent of ambiguity in natural language is explored systematically by Church and Patil (1982), who discuss techniques such as canonical forms and packing in detail. D-theory is described in Marcus et al. (1983) and has been formalized for use within the TAG formalism by Vijay-Shankar (1992). Fidditch (Hindle, 1989) is one of

the best examples of a parser that only analyzes structure that can be reliably identified. Its output is a set of fragments rather than a complete interpretation.

>> [back](#)

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 184]

Exercises for Chapter 6

1. (easy) Give a trace of the shift-reduce parser using the oracle in Figure 6.6 as it parses the sentence *The dog was eating the bone*.
2. (medium) List and define the three parsing strategies that this text says people use. Discuss how the following sentences are ambiguous, and state which reading is preferred in the framework of these parsing strategies.
 - a. **It flew past the geese over the field.**
 - b. **The artist paints the scene in the park.**
 - c. **He feels the pain in his heart.**
3. (medium) Build an oracle for the following grammar:

S → NP VP

NP → ART N

NP → ART N PP

VP → V NP

S → NP VP

VP → V NP PP

Identify places where there is an ambiguity that the technique described cannot resolve, and use the heuristics at the end of Section 6.2 to select an unambiguous action for the oracle. Give an example of a sentence that will be incorrectly parsed because of this decision.

4. (medium)

- a. Construct the oracle for the following VP grammar involving to-infinitives and prepositions:

VP → V NP

VP → V INF

VP → V PP

INF → to VP

PP → P NP

NP → DET N

NP → NAME

b. Can the oracle correctly analyze both of the following VPs without resorting to guessing? Trace the parse for each sentence starting at the word "walked".

Jack walked to raise the money.

Jack walked to the store.

c. Consider allowing word ambiguity in the grammar. In particular, the word *sand* can be either a noun or a verb. What does your parser do with the VPs

Jack turned to sand.

Jack turned to sand the board.

[Allen 1995: Chapter 6 - Toward Efficient Parsing / 185]

5. (medium) Design a deterministic grammar that can successfully parse both

I know that boy is lazy
 | | |

I know **that boy** is lazy

NP V ADJ

and

I know **that** boys are lazy

| | | |

| NP V ADJ

COMPLEMENTIZER

You may use as much of the grammar presented in this chapter as you wish. Trace the parser on each of these sentences in the style found in this chapter. Describe the important points of your analysis. Discuss how general your solution is in dealing with the various uses of the word *that*. Show at least one further example of a sentence involving *that* and outline how your grammar accounts for it (them).

6. (medium) Modify an existing chart parser so that it uses a packed chart, as described in Section 6.4. Perform a set of experiments with a grammar without features and compare the size of the chart with and without packing.

7. (medium) Construct an example that shows what problems arise when you use a packed chart with a grammar with features. Devise a strategy for dealing with the problems.

8. (medium) Assume a chart parser using a packed representation. Develop an upper limit on the number of operations the parser will have to do to parse a sentence of length S using a grammar that uses N nonterminals and T terminal symbols.

>> [back](#)

Allen 1995 : Natural Language Understanding

	<u>Contents</u>	<u>Preface</u>	<u>Introduction</u>	
<u>previous chapter</u>	<u>Part I Syntactic Processing</u>	<u>Part II Semantic Interpretation</u>	<u>Part III - Context / World Knowledge</u>	<u>next chapter</u>
	<u>Appendices</u>	<u>Bibliography</u>		
	<u>Summaries</u>	<u>Further Readings</u>	<u>Exercises</u>	

Chapter 7 - Ambiguity Resolution: Statistical Methods

	<u>7.1 Basic Probability Theory</u>
	<u>7.2 Estimating Probabilities</u>
	<u>7.3 Part-of-Speech Tagging</u>
	<u>7.4 Obtaining Lexical Probabilities</u>
	<u>7.5 Probabilistic Context-Free Grammars</u>
	<u>7.6 Best-First Parsing</u>
	<u>7.7 A Simple Context-Dependent Best-First Parser</u>
	<u>Summary</u>

	<u>7.1 Basic Probability Theory</u>
	<u>Related Work and Further Readings</u>
	<u>Exercises for Chapter 7</u>

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 189]

Chapter 6 suggested employing heuristics to choose between alternate syntactic analyses. Creating such heuristics, however, is difficult and time consuming. Furthermore, there is no systematic method for evaluating how well the heuristic rules work in practice. This section explores some techniques for solving these problems based on probability theory. Such approaches have become very popular in the past few years because large databases, or corpora, of natural language data have become available. This data allows you to use statistically based techniques for automatically deriving the probabilities needed. The most commonly available corpus, the Brown corpus, consists of about a million words, all labeled with their parts of speech. More recently, much larger

databases have become available in formats ranging from raw text format (as in databases of articles from the AP news wire and the "*Wall Street Journal*") to corpora with full syntactic annotation (such as the Penn Treebank). The availability of all this data suggests new analysis techniques that were previously not possible.

Section 7.1 introduces the key ideas in probability theory and explores how they might apply to natural language applications. Section 7.2 considers some techniques for estimating probabilities from corpora, and Section 7.3 develops techniques for part-of-speech tagging. Section 7.4 defines the technique for computing lexical probabilities. Section 7.5 introduces probabilistic context-free grammars, and Section 7.6 explores the issue of building a probabilistically driven best-first parser. The final section introduces a context-dependent probability estimate that has significant advantages over the context-free method.

>> [back](#)

7.1 Basic Probability Theory

Intuitively, the probability of some event is the likelihood that it will occur. A probability of 1 indicates that the event is certain to occur, while a probability of 0 indicates that the event definitely will not occur. Any number between 0 and 1 indicates some degree of uncertainty. This uncertainty can be illuminated by considering the odds of the event occurring, as you would if you were going to bet on whether the event will occur or not. A probability of .5 would indicate that the event is equally likely to occur as not to occur, that is, a "50/50" bet. An event with probability .5 would occur exactly half of the time. An event with probability .1 would occur once in every 10 opportunities (1/10 odds), whereas an event with probability .75 would occur 75 times out of 100 (3/4 odds).

More formally, probability can be defined in terms of a **random variable**, which may range over a predefined set of values. While random variables may range over infinite sets and continuous values, here we will use only random variables that range over a finite set of values. For instance, consider tossing a coin. The random variable TOSS representing the result of a coin toss would range over two possible values: heads (h) or tails (t). One possible event is that the coin comes up heads - TOSS=h; the other is that the coin comes up tails - TOSS=t. No other value for TOSS is possible, reflecting the fact that a tossed coin always comes up either heads or tails. Usually, we will not mention the

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 190]

random variable and will talk of the probability of TOSS=h simply as the probability of the event h.

A probability function, **PROB**, assigns a probability to every possible value of a random variable. Every probability function must have the following properties, where e_1, \dots, e_n are the possible distinct values of a random variable E:

1. **PROB(e_i) ≤ 0 , for all i**
2. **PROB(e_i) ≥ 1 , for all i**
3. **$\sum_{i=1}^n \text{PROB}(e_i) = 1$**

Consider a specific example. A particular horse, Harry, ran 100 races in his career. The result of a race is represented as a random variable R that has one of two values, Win or Lose. Say that Harry won 20 times overall. Thus the probability of Harry winning the race is **PROB(R=Win) = .2** and the probability of him losing is **PROB(R=Lose) = .8**. Note that these values satisfy the three constraints just defined.

Of course, there may be many different random variables, and we are often interested in how they relate to each other. Continuing the racing example, consider another random variable W representing the state of the weather and ranging over the values Rain and Shine. Let us say that it was raining 30 times out of the 100 times a race was run. Of these 30 races, Harry won 15 times. Intuitively, if you were given the fact that it was raining - that is, that $W=\text{Rain}$ - the probability that $R=\text{Win}$ would be .5 (15 out of 30). This intuition is captured by the concept of conditional probability and is written as **PROB**(Win | Rain), which is often described as the probability of the event Win given the event Rain. Conditional probability is defined by the formula

$$\text{PROB}(e \mid e') = \frac{\text{PROB}(e \text{ & } e')}{\text{PROB}(e')}$$

where **PROB**(e & e') is the probability of the two events e and e' occurring simultaneously.

You know that **PROB**(Rain) = .3 and **PROB**(Win & Rain) = .15, and using the definition of conditional probability you can compute **PROB**(Win | Rain) and see that it agrees with your intuition:

$$\begin{aligned} \text{PROB}(\text{Win} \mid \text{Rain}) &= \frac{\text{PROB}(\text{Win} \text{ & } \text{Rain})}{\text{PROB}(\text{Rain})} \\ &= \frac{.15}{.30} \\ &= .5 \end{aligned}$$

An important theorem relating conditional probabilities is called **Bayes' rule**. This rule relates the conditional probability of an event A given B to the conditional probability of B given A:

$$\frac{\text{PROB}(B \mid A) * \text{PROB}(A)}{\text{PROB}(A \mid B)}$$

$$\frac{\text{PROB}(B)}{\text{PROB}(B)}$$

We can illustrate this rule using the race horse example. Using Bayes' rule we can compute the probability that it rained on a day that Harry won a race:

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 191]

$$\begin{aligned}\text{PROB}(\text{Rain} \mid \text{Win}) &= (\text{PROB}(\text{Win} \mid \text{Rain}) * \text{PROB}(\text{Rain})) / \text{PROB}(\text{Win}) \\ &= (.5 * .3) / .2 \\ &= .75\end{aligned}$$

which, of course, is the same value as if we calculated the conditional probability directly from its definition:

$$\begin{aligned}\text{PROB}(\text{Rain} \mid \text{Win}) &= (\text{PROB}(\text{Rain} \& \text{Win}) * \text{PROB}(\text{Win})) \\ &= .15 / .20 \\ &= .75\end{aligned}$$

The reason that Bayes' rule is useful is that we usually do not have complete information about a situation and so do not know all the required probabilities. We can, however, often estimate some probabilities reasonably and then use Bayes' rule to calculate the others.

Another important concept in probability theory is the notion of independence. Two events are said to be independent of each other if the occurrence of one does not affect the probability of the occurrence of the other. For instance, in the race horse example, consider another random variable, L, that indicates whether I took my lucky rabbit's foot to the race (value Foot) or not (Empty). Say I took my rabbit's foot 60 times, and Harry won 12 races. This means that the probability that Harry won the race on a day that I took the rabbit's foot is $\text{PROB}(\text{Win} \mid \text{Foot}) = 12/60 = .2$. Since this is the same as the usual probability of Harry winning, you can conclude that winning the race is independent of taking the rabbit's foot.

More formally, two events A and B are independent of each other if and only if

$$\text{PROB}(A \mid B) = \text{PROB}(A)$$

which, using the definition of conditional probability, is equivalent to saying

$$\text{PROB}(A \ \& \ B) = \text{PROB}(A) * \text{PROB}(B)$$

Note that the events of winning and raining are not independent, given that $\text{PROB}(\text{Win} \ \& \ \text{Rain}) = .15$ while $\text{PROB}(\text{Win}) * \text{PROB}(\text{Rain}) = .2 * .3 = .06$. In other words, winning and raining occur together at a rate much greater than random chance.

Consider an application of probability theory related to language, namely part-of-speech identification: Given a sentence with ambiguous words, determine the most likely lexical category for each word. This problem will be examined in detail in Section 7.3. Here we consider a trivial case to illustrate the ideas underlying probability theory. Say you need to identify the correct syntactic category for words that can be either nouns or verbs. This can be formalized using two random variables: one, C, that ranges over the parts of speech (N and V), and another, W, that ranges over all the possible words. Consider an example when W = "*flies*". The problem can be stated as determining whether $\text{PROB}(C=N | W=\text{"flies"})$ or $\text{PROB}(C=V | W=\text{"flies"})$ is greater. Note that, as mentioned earlier, the

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 192]

random variables will usually be omitted from the formulas. Thus $\text{PROB}(C=N \mid W=\text{"flies"})$ will usually be abbreviated as $\text{PROB}(N \mid \text{"flies"})$. Given the definition of conditional probability, the probabilities for the categories of the word *flies* are calculated as follows:

$$\text{PROB}(N \mid \text{"flies"}) = \text{PROB}(\text{"flies"} \ \& \ N) / \text{PROB}(\text{"flies"})$$

$$\text{PROB}(V \mid \text{"flies"}) = \text{PROB}(\text{"flies"} \ \& \ V) / \text{PROB}(\text{"flies"})$$

So the problem reduces to finding which of $\text{PROB}(\text{"flies"} \ \& \ N)$ and $\text{PROB}(\text{"flies"} \ \& \ V)$ is greater, since the denominator is the same in each formula.

How might you obtain these probabilities? You clearly cannot determine the true probabilities since you don't have a record of all text ever written, let alone the text that will be processed in the future. But if you have a large enough sample of data, you can estimate the probabilities.

Let's say we have a corpus of simple sentences containing 1,273,000 words. Say we find 1000 uses of the word "flies", 400 of them in the N sense, and 600 in the V sense. We can approximate the probabilities by looking at the ratios of the frequencies of the words in the corpus. For example, the probability of a randomly selected word being the word "flies" is

$$\text{PROB}(\text{"flies"}) \approx 1000 / 1,273,000 = 0.0008$$

The joint probabilities for "flies" as a noun and "flies" as a verb are

PROB(*flies* & N) Z 400 / 1.273.000 = 0.0003

PROB(*flies* & V) Z 600 / 1.273.000 = 0.0005

Thus our best guess is that each use of "*flies*" is a verb. We can compute the probability that "*flies*" is a verb using the formula

$$\begin{aligned}\text{PROB (V | } \textit{flies}) &= \text{PROB(V \& flies)} / \text{PROB (flies)} \\ &= .0005 / .0008 = .625\end{aligned}$$

Using this method, an algorithm that always asserts "*flies*" to be a verb will be correct about 60 percent of the time. This is clearly a poor strategy, but better than guessing that it is a noun all the time! To get a better method, you must consider more context, such as the sentence in which "*flies*" occurs. This idea is developed in Section 7.3.

>> [back](#)

7.2 Estimating Probabilities

If you have all the data that would ever be relevant to a problem, you can compute exact probabilities for that data. For instance, say Harry, the horse in the last section, ran only 100 races and then was put out to pasture. Now you can compute the exact probability of Harry winning any particular race someone might choose of the 100 possibilities. But, of course, this is not how probability is generally used. Typically, you want to use probability to predict future behavior; that is, you'd use information on Harry's past performance to predict how likely he is

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 193]

Results	Estimate of <i>Prob(H)</i>	Acceptable Estimate?
HH	1.0	NO
HT	.5	YES
TH	.5	YES
TT	0	NO

Figure 7.1 Possible estimates of *Prob(H)* given two flips

Figure 7.1 Possible estimates of *Prob(H)* given two flips

to win his 101st race (which you are going to bet on). This is a real-life application of probability theory. You are in the same position when using probabilities to help resolve ambiguity, since you are interested in parsing sentences that have never been seen before. Thus you need to use data on previously occurring sentences to predict the interpretation of the next sentence. We will always be working with estimates of probability rather than the actual probabilities.

As seen in the last section, one method of estimation is to use the ratios from the corpus as the probability to predict the interpretation of the new sentence. Thus, if we have seen the word "flies" 1000 times before, and 600

of them were as a verb, then we assume that $\text{PROB}(V \mid "flies")$ is .6 and use that to guide our guess with the 1001st case. This simple ratio estimate is called the **maximum likelihood estimator (MLE)**. If you have many examples of the events you are estimating, this can be a very reliable estimate of the true probability.

In general, the accuracy of an estimate increases as the amount of data used expands, and there is a theorem of statistics - the law of large numbers - that states that estimates can be made as accurate as desired if you have unlimited data. Estimates can be particularly unreliable, however, when only a small number of samples are involved. Consider trying to estimate the true probability of a fair coin coming up heads when it is flipped. For the sake of discussion, since we know the actual answer is .5, let us say the estimate is accurate enough if it falls between .25 and .75. This range will be called the acceptable margin of error. If you only do two trials, there are four possible outcomes, as shown in Figure 7.1: two heads, heads then tails, tails then heads, or two tails. This means that, if you flip a coin twice, half the time you will obtain an estimate of 1/2 or .5. The other half of the time, however, you will estimate that the probability of coming up heads is 1 (the two-heads case) or 0 (the two-tails case). So you have only a 50 percent chance of obtaining an estimate within the desired margin of error. With three flips, the chance of getting a reliable enough estimate jumps to 75 percent, as shown in Figure 7.2. With four flips, there is an 87.5 percent chance of the estimate being accurate enough, with eight flips a 93 percent chance, with 12 flips a 95 percent chance, and so on. No matter how long you flip, there is always a chance that the estimate found is inaccurate, but you can reduce the probability of this occurring to as small a number as you desire if you can do enough trials.

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 194]

Results	Estimate of <i>Prob(H)</i>	Acceptable Estimate?
HHH	1.0	NO
HHT	.66	YES
HTH	.66	YES
HTT	.33	YES
THH	.66	YES
THT	.33	YES
TTH	.33	YES
TTT	0	NO

Figure 7.2 Possible estimates of *Prob(H)* given three flips

Figure 7.2 Possible estimates of *Prob(H)* given three flips

Almost any method of estimation works well when there is a lot of data. Unfortunately, there are a vast number of estimates needed for natural language applications, and a large proportion of these events are quite rare. This is the problem of sparse data. For instance, the Brown corpus contains about a million words, but due to duplication there are only 49,000 different words. Given this, you might expect each word to occur about 20 times on average. But over 40,000 of the words occur five times or less. With such few numbers, our estimates of the part of speech for such words may be highly inaccurate. The worst case occurs if a low-frequency word does not occur at all in one of its possible categories. Its probability in this category would then be estimated as 0; thus no interpretation using the word in any category would be possible, because the probability of the overall sentence containing the word would be 0 as well. Unfortunately, rare events are common enough in natural language applications that reliable estimates for these low-frequency words are essential for the algorithms.

There are other estimation techniques that attempt to address the problem of estimating probabilities of low-frequency events. To examine them, let's introduce a framework in which they can be compared. For a particular random variable X , all techniques start with a set of values, \mathbf{v}_i , computed from the count of the number of times $X = x_i$. The maximum likelihood estimation technique uses $V_1 = |x_i|$; that is, V_i is exactly the count of the number of times $X = x_i$. Once V_i is determined for each x_i , the probability estimates are obtained by the formula

$$\text{PROB } (X = x_i) \approx v_i / \sum_i v_i$$

Note that the denominator guarantees that the estimates obey the three properties of a probability function defined in Section 7.1.

One technique to solve the zero probability problem is to ensure that no V_i has the value 0. We might, for instance, add a small amount, say .5, to every count, such as in $V_i = |x_i| + .5$. This guarantees no zero probabilities

yet retains

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 195]

the relative likelihoods for the frequently occurring values. This estimation technique is called the **expected likelihood estimator (ELE)**. To see the difference between this and the MLE, consider a word w that happens not to occur in the corpus, and consider estimating the probability that w occurs in one of 40 word classes $L_1 \dots L_{40}$. Thus we have a random variable X , where $X = x_i$ only if w appears in word category L_i . The MLE for $\text{PROB}(X = x_i)$ will not be defined because the formula has a zero denominator. The ELE, however, gives an equally likely probability to each possible word class. With 40 word classes, for instance, each V_i will be .5, and thus $\text{PROB}(L_1 | w) \approx .5/20 = .025$. This estimate better reflects the fact that we have no information about the word. On the other hand, the ELE is very conservative. If w appears in the corpus five times, once as a verb and four times as a noun, then the MLE estimate of $\text{PROB}(N | w)$ would be .8, while the ELE estimate would be $4.5/25 = .18$, a very small value compared to intuition.

>> [back](#)

Evaluation

Once you have a set of estimated probabilities and an algorithm for some particular application, you would like to be able to tell how well your new technique performs compared with other algorithms or variants of your algorithm. The general method for doing this is to divide the corpus into two parts: the training set and the test set. Typically, the test set consists of 10 - 20 percent of the total data. The training set is then used to estimate the probabilities, and the algorithm is run on the test set to see how well it does on new data. Running the algorithm on the training set is not considered a reliable method of evaluation because it does not measure the generality of your technique. For instance, you could do well on the training set simply by remembering all the answers and repeating them back in the test! A more thorough method of testing is called cross-validation, which involves repeatedly removing different parts of the corpus as the test set, training on the remainder of the corpus, and then evaluating on the new test set. This technique reduces the chance that the test set selected was somehow easier than you might expect.

>> [back](#)

7.3 Part-of-Speech Tagging

Part-of-speech tagging involves selecting the most likely sequence of syntactic categories for the words in a sentence. A typical set of tags, used in the Penn Treebank project, is shown in Figure 7.3. In Section 7.1 you saw the simplest algorithm for this task: Always choose the interpretation that occurs most frequently in the training set. Surprisingly, this technique often obtains about a 90 percent success rate, primarily because over half the words appearing in most corpora are not ambiguous. So this measure is the starting point from which to evaluate algorithms that use more sophisticated techniques. Unless a method does significantly better than 90 percent, it is not working very well.

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 196]

1.	CC	Coordinating conjunction	19.	PP\$	Possessive pronoun
----	----	--------------------------	-----	------	--------------------

1.	CC	Coordinating conjunction	19.	PP\$	Possessive pronoun
2.	CD	Cardinal number	20.	RB	Adverb
3.	DL'	Determiner	21.	RBR	Comparative adverb
4.	EX	Existential <i>there</i>	22.	RBS	Superlative Adverb
5.	FW	Foreign word	23.	RP	Particle
6.	IN	Preposition / subord. conj	24.	SYM	Symbol (math or scientific)
7.	11	Adjective	25.	10	to
8.	JJR	Comparative adjective	26.	UH	Interjection
9.	JJS	Superlative adjective	27.	VB	Verb, base form
10.	LS	List item marker	28.	VBD	Verb, past tense

1.	CC	Coordinating conjunction	19.	PP\$	Possessive pronoun
11.	MD	Modal	29.	VBG	Verb, gerund/pres. participle
12.	NN	Noun, singular or mass	30.	VBN	Verb, past participle
13.	NNS	Noun, plural	31.	VBP	Verb, non-3s, present
14.	NNP	Proper noun, singular	32.	VBZ	Verb, 3s, present
15.	NNPS	Proper noun, plural	33.	WDT	Wh-determiner
16.	PDT	Predeterminer	34.	WP	Wh-pronoun
17.	POS	Possessive ending	35.	WPZ	Possessive wh-pronoun
18.	PRP	Personal pronoun	36.	WRB	Wh-adverb

Figure 7.3 The Penn Treebank tagset

The general method to improve reliability is to use some of the local context of the sentence in which the word appears. For example, in Section 7.1 you saw that choosing the verb sense of "flies" in the sample corpus was the best choice and would be right about 60 percent of the time. If the word is preceded by the word "*the*", on the other hand, it is much more likely to be a noun. The technique developed in this section is able to exploit such information.

Consider the problem in its full generality. Let w_1, \dots, w_T be a sequence of words. We want to find the sequence of lexical categories C_1, \dots, C_T that maximizes

$$1. \text{ PROB } (C_1, \dots, C_T \mid w_1, \dots, w_T)$$

Unfortunately, it would take far too much data to generate reasonable estimates for such sequences, so direct methods cannot be applied. There are, however, reasonable approximation techniques that produce good results. To develop them, you must restate the problem using Bayes' rule, which says that this conditional probability equals

$$2. \frac{\text{PROB } (C_1, \dots, C_T) * \text{PROB } (W_1, \dots, W_T | C_1, \dots, C_T)}{\text{PROB } (W_1, \dots, W_T)}$$

As before, since we are interested in finding the C_1, \dots, C_n that gives the maximum value, the common denominator in all these cases will not affect the answer. Thus the problem reduces to finding the sequence C_1, \dots, C_n that maximizes the formula

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 197]

$$3. \text{PROB } (C_1, \dots, C_T) * \text{PROB } (W_1, \dots, W_T | C_1, \dots, C_T)$$

There are still no effective methods for calculating the probability of these long sequences accurately, as it would require far too much data. But the probabilities can be approximated by probabilities that are simpler to collect by making some independence assumptions. While these independence assumptions are not really valid, the estimates appear to work reasonably well in practice. Each of the two expressions in formula 3 will be approximated. The first expression, the probability of the sequence of categories, can be approximated by a series of probabilities based on a limited number of previous categories. The most common assumptions use either one or two previous categories. The bigram model looks at pairs of categories (or words) and uses the conditional probability that a category C_1 will follow a category C_{i-1} , written as $\text{PROB}(C_1 | C_{i-1})$. The **trigram** model uses the conditional probability of one category (or word) given the two preceding categories (or words), that is, $\text{PROB}(C_i | C_{i-1}, C_{i-2})$.

$| C_{i-2} C_{i-1})$. These models are called **n-gram** models, in which n represents the number of words used in the pattern. While the trigram model will produce better results in practice, we will consider the bigram model here for simplicity. Using bigrams, the following approximation can be used:

$$\text{PROB } (C_1, \dots, C_T) \approx \prod_{i=1, T} \text{PROB}(C_i | C_{i-1})$$

To account for the beginning of a sentence, we posit a pseudocategory \ddot{Y} at position 0 as the value of C_0 . Thus the first bigram for a sentence beginning with an ART would be $\text{PROB}(\text{ART} | \ddot{Y})$. Given this, the approximation of the probability of the sequence ART N V N using bigrams would be

$$\begin{aligned} \text{PROB } (\text{ART } N \ V \ N) &\approx \text{PROB } (\text{ART} | \ddot{Y}) * \text{PROB } (N | \text{ART}) \\ &\quad * \text{PROB } (V | N) * \text{PROB } (N | V) \end{aligned}$$

The second probability in formula 3,

$$\text{PROB } (W_1, \dots, W_T | C_1, \dots, C_T)$$

can be approximated by assuming that a word appears in a category independent of the words in the preceding or succeeding categories. It is approximated by the product of the probability that each word occurs in the indicated

part of speech. that is, by

$$\text{PROB } (w_1, \dots, w_T \mid c_1, \dots, c_T) \approx \prod_{i=1}^T \text{PROB}(w_i \mid c_i)$$

With these two approximations, the problem has changed into finding the sequence c_1, \dots, c_T that maximizes the value of

$$\prod_{i=1}^T \text{PROB}(c_i \mid c_{i-1}) * \text{PROB } \text{PROB}(w_i \mid c_i)$$

The advantage of this new formula is that the probabilities involved can be readily estimated from a corpus of text labeled with parts of speech. In particular, given a database of text, the bigram probabilities can be estimated simply by counting the number of times each pair of categories occurs compared to the

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 198]

Category	Count at i	Pair	Count at i,i+1	Bigram	Estimate
\emptyset	300	\emptyset, ART	213	$PROB(\text{ART} \emptyset)$.71
\emptyset	300	\emptyset, N	87	$PROB(\text{N} \emptyset)$.29
ART	558	ART, N	558	$PROB(\text{N} \text{ART})$	1
N	833	N, V	358	$PROB(\text{V} \text{N})$.43
N	833	N, N	108	$PROB(\text{N} \text{N})$.13
N	833	N, P	366	$PROB(\text{P} \text{N})$.44
V	300	V, N	75	$PROB(\text{N} \text{V})$.35
V	300	V, ART	194	$PROB(\text{ART} \text{V})$.65
P	307	P, ART	226	$PROB(\text{ART} \text{P})$.74
P	307	P, N	81	$PROB(\text{N} \text{P})$.26

Figure 7.4 Bigram probabilities from the generated corpus

Figure 7.4 Bigram probabilities from the generated corpus

individual category counts. The probability that a V follows an N would be estimated as follows:

$$\text{PROB}(C_i = V \mid C_{i-1} = N) Z = \frac{\text{Count}(N \text{ at position } i-1 \text{ and } V \text{ at } i)}{\text{Count}(N \text{ at position } i-1)}$$

Figure 7.4 gives some bigram frequencies computed from an artificially generated corpus of simple sentences. The corpus consists of 300 sentences but has words in only four categories: N, V, ART, and P. In contrast, a typical real tagset used in the Penn Treebank, shown in Figure 7.3, contains about 40 tags. The artificial corpus contains 1998 words: 833 nouns, 300 verbs, 558 articles, and 307 prepositions. Each bigram is estimated using the previous formula. To deal with the problem of sparse data, any bigram that is not listed here will be assumed to have a token probability of .0001.

The lexical-generation probabilities, $\text{PROB}(w_i \mid C_1)$, can be estimated simply by counting the number of occurrences of each word by category. Figure 7.5 gives some counts for individual words from which the lexical-generation probability estimates in Figure 7.6 are computed. Note that the lexical-generation probability is the probability that a given category is realized by a specific word, not the probability that a given word falls in a specific category. For instance, $\text{PROB}(the \mid \text{ART})$ is estimated by $\text{Count}(\# \text{ times } the \text{ is an ART}) / \text{Count}(\# \text{ times an ART occurs})$. The other probability, $\text{PROB}(\text{ART} < the)$, would give a very different value.

Given all these probability estimates, how might you find the sequence of categories that has the highest probability of generating a specific sentence? The brute force method would be to generate all possible sequences that could generate the sentence and then estimate the probability of each and pick the best one. The problem with this is that there are an exponential number of sequences - given N categories and T words, there are NT possible sequences.

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 199]

	N	V	ART	P	TOTAL
<i>flies</i>	21	23	0	0	44
<i>fruit</i>	49	5	1	0	55
<i>like</i>	10	30	0	21	61
<i>a</i>	1	0	201	0	202

	N	V	ART	P	TOTAL
<i>the</i>	1	0	300	2	303
<i>flower</i>	53	15	0	0	68
<i>flowers</i>	42	16	0	0	58
<i>birds</i>	64	1	0	0	65
others	592	210	56	284	1142
TOTAL	833	300	558	307	1998

Figure 7.5 A summary of some of the word counts in the corpus

PROB (<i>the</i> ART)	.54	PROB (<i>a</i> ART)	.360
PROB (<i>flies</i> N)	.025	PROB (<i>a</i> N)	.001
PROB (<i>flies</i> V)	.076	PROB (<i>flower</i> N)	.063
PROB (<i>like</i> V)	.1	PROB (<i>flower</i> V)	.05
PROB (<i>like</i> P)	.068	PROB (<i>birds</i> N)	.076
PROB (<i>like</i> N)	.012		

Figure 7.6 The lexical-generation probabilities

Luckily, you can do much better than this because of the independence assumptions that were made about the data.

Since we are only dealing with bigram probabilities, the probability that the i^{th} word is in a category C_1 depends only on the category of the $(i-1)^{\text{th}}$ word, C_{i-1} . Thus the process can be modeled by a special form of probabilistic finite state machine, as shown in Figure 7.7. Each node represents a possible lexical category and the transition probabilities (the bigram probabilities in Figure 7.4) indicate the probability of one category following another.

With such a network you can compute the probability of any sequence of categories simply by finding the path through the network indicated by the sequence and multiplying the transition probabilities together. For instance, the sequence ART N V N would have the probability $.71 * 1 * .43 * .35 = .107$. This representation, of course, is only accurate if the probability of a category occurring depends only on the one category before it. In probability theory this is often called the Markov assumption, and networks like that in Figure 7.7 are called Markov chains.

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 200]

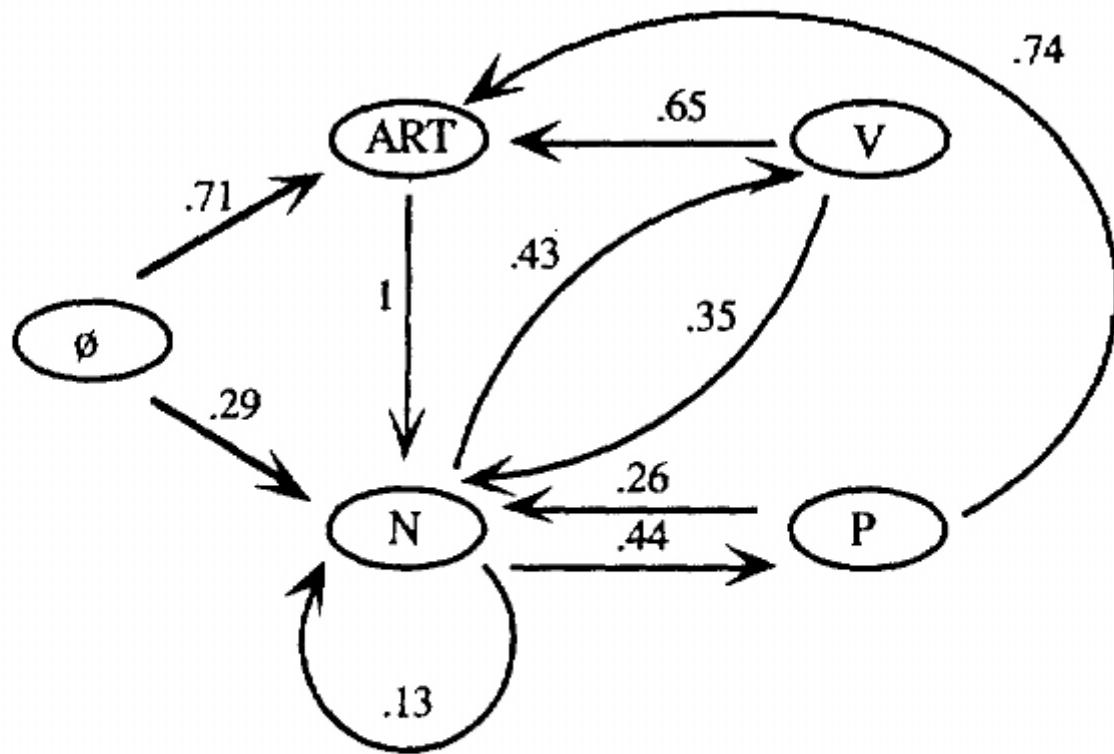


Figure 7.7 A Markov chain capturing the bigram probabilities

Figure 7.7 A Markov chain capturing the bigrain probabilities

The network representation can now be extended to include the lexical-generation probabilities, as well. In particular, we allow each node to have an output probability, which gives a probability to each possible output that could correspond to the node. For instance, node N in Figure 7.7 would be associated with a probability table that indicates, for each word, how likely that word is to be selected if we randomly select a noun. The output probabilities are exactly the lexical-generation probabilities shown in Figure 7.6. A network like that in Figure 7.7 with output probabilities associated with each node is called a **Hidden Markov Model (HMM)**. The word "hidden" in the name indicates that for a specific sequence of words, it is not clear what state the Markov model is in. For instance, the word "*flies*" could be generated from state N with a probability of .025 (given the values in Figure 7.6), or it could be generated from state V with a probability .076. Because of this ambiguity, it is no longer trivial to compute the probability of a sequence of words from the network. If you are given a particular sequence, however, the probability that it generates a particular output is easily computed by multiplying the probabilities on the path times the probabilities for each output. For instance, the probability that the sequence N V ART N generates the output "*Flies like a flower*" is computed as follows. The probability of the path N V ART N, given the Markov model in Figure 7.7, is $.29 * .43 * .65 * 1 = .081$. The probability of the output being "*Flies like a flower*" for this sequence is computed from the output probabilities given in Figure 7.6:

```

PROB(flies | N) * PROB (like | V) * PROB(a | ART) * PROB
(flower | N)

= .025 * .1 * .36 * .063

= 5.4 * 10-5

```

Multiplying these together gives us the likelihood that the HMM would generate the sentence, $4.37 * 10^{-6}$. More generally, the formula for computing the probability of a sentence w_1, \dots, w_T given a sequence C_1, \dots, C_T is

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 201]

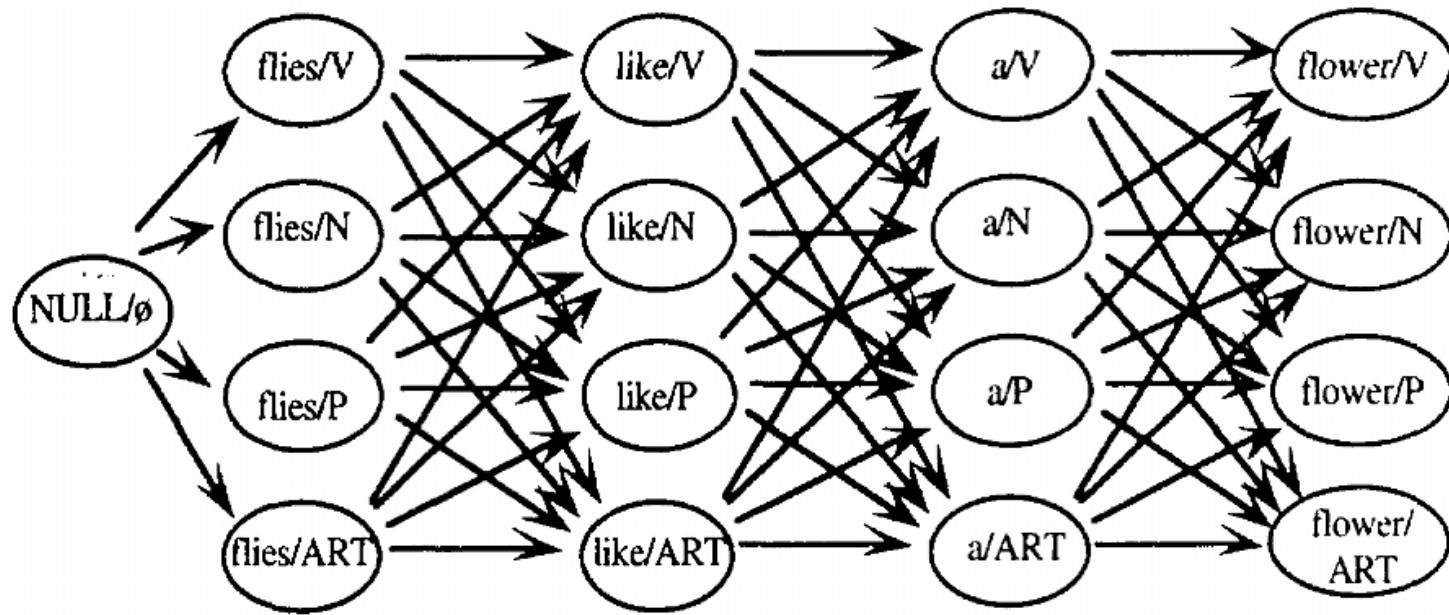


Figure 7.8 Encoding the 256 possible sequences exploiting the Markov assumption

Figure 7.8 Encoding the 256 possible sequences exploiting the Markov assumption

$$\langle_{i=1, T} \text{PROB}(C_i | C_{i-1}) * \text{PROB}(w_i | C_i)$$

Now we can resume the discussion of how to find the most likely sequence of categories for a sequence of words. The key insight is that because of the Markov assumption, you do not have to enumerate all the possible sequences. In fact, sequences that end in the same category can be collapsed together since the next category only depends on the one previous category in the sequence. So if you just keep track of the most likely sequence found so far for each possible ending category, you can ignore all the other less likely sequences. For example. consider the problem of finding the most likely categories for the sentence "*Flies like a flower*", with the lexical-generation probabilities and bigram probabilities discussed so far. Given that there are four possible categories, there are $4^4 = 256$ different sequences of length four. The brute force algorithm would have to generate all 256 sequences and compare their probabilities in order to find this one. Exploiting the Markov assumption, however, this set of sequences can be collapsed into a representation that considers only the four possibilities for each word. This representation, shown as a transition diagram in Figure 7.8, represents all 256 sequences. To find the most likely sequence, you sweep forward through the words one at a time finding the most likely sequence for each ending category. In other words, you first find the four best sequences for the two words "*flies like*": the best ending with "*like*" as a V, the best as an N, the best as a P, and the best as an ART. You then use this information to find the four best sequences for the three words "*flies like a*", each one ending in a different category. This process is repeated until all the words are accounted for. This algorithm is usually called the **Viterbi** algorithm. For a problem involving T words and N lexical categories, it is guaranteed to find the most likely sequence using $k*T*N^2$ steps, for some constant k , significantly better than the N^T steps required by the brute force search! The rest of this section develops the Viterbi algorithm in detail.

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 201]

Given word sequence w_1, \dots, w_T , lexical categories L_1, \dots, L_N , lexical probabilities $PROB(w_t | L_i)$, and bigram probabilities $PROB(L_i | L_j)$, find the most likely sequence of lexical categories C_1, \dots, C_T for the word sequence.

Initialization Step

For $i = 1$ to N do

$$\text{SEQSCORE}(i, 1) = PROB(w_1 | L_i) * PROB(L_i | \emptyset)$$

$$\text{BACKPTR}(i, 1) = 0$$

Iteration Step

For $t = 2$ to T

 For $i = 1$ to N

$$\text{SEQSCORE}(i, t) = \max_{j=1,N} (\text{SEQSCORE}(j, t-1) * PROB(L_i | L_j)) * PROB(w_t | L_i)$$

$\text{BACKPTR}(i, t) = \text{index of } j \text{ that gave the max above}$

Sequence Identification Step

$C(T) = i$ that maximizes $\text{SEQSCORE}(i, T)$

For $i = T-1$ to 1 do

$$C(i) = \text{BACKPTR}(C(i+1), i+1)$$

Figure 7.9 The Viterbi algorithm

Figure 7.9 The Viterbi algorithm

o The Viterbi Algorithm

We will track the probability of the best sequence leading to each possible category at each position using an $N \times T$ array, where N is the number of lexical categories (L_1, \dots, L_N) and T is the number of words in the sentence (w_1, \dots, w_T). This array, $\text{SEQSCORE}(n, t)$, records the probability for the best sequence up to position t that ends with a word in category L_n . To record the actual best sequence for each category at each position, it suffices to record only the one preceding category for each category and position. Another $N \times T$ array, BACKPTR , will indicate for each category in each position what the preceding category is in the best sequence at position $t-1$. The algorithm, shown in Figure 7.9, operates by computing the values for these two arrays.

Let's assume you have analyzed a corpus and obtained the bigram and lexical-generation probabilities in Figures 7.4 and 7.6, and assume that any bigram probability not in Figure 7.4 has a value of .0001. Using these probabilities, the algorithm running on the sentence "*Flies like a flower*" will operate as follows.

The first row is set in the initialization phase using the formula

```
SEQSCORE (i, 1) = PROB (flies | Li) * PROB (Li | Y)
```

where L_1 ranges over V, N, ART, and P. Because of the lexical-generation probabilities in Figure 7.6. only the entries for a noun and a verb are greater than zero.

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 203]

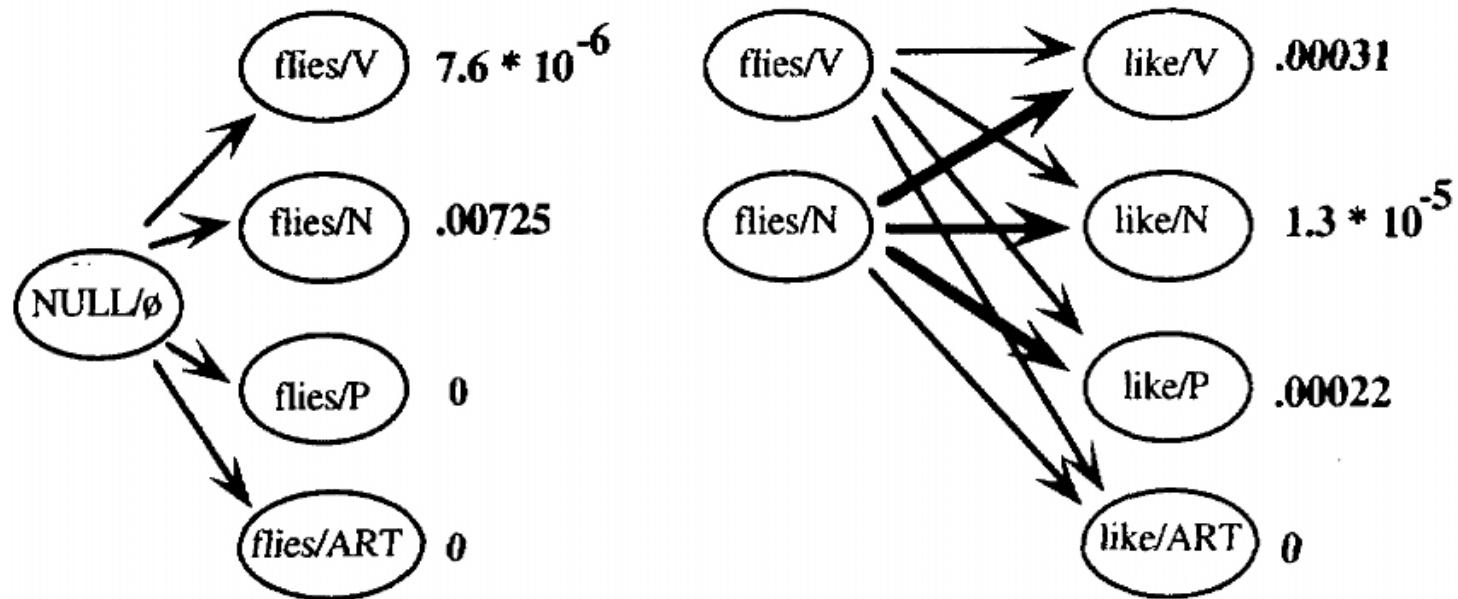


Figure 7.10 The results of the first two steps of the Viterbi algorithm

Figure 7.10 The results of the first two steps of the Viterbi algorithm

Thus the most likely sequence of one category ending in a V (= L₁) to generate "flies" has a score of 7.6×10^{-6} , whereas the most likely one ending in an N (= L₂) has a score of .00725.

The result of the first step of the algorithm is shown as the left-hand side network in Figure 7.10. The probability of "flies" in each category has been computed. The second phase of the algorithm extends the sequences one word at a time, keeping track of the best sequence found so far to each category. For instance, the probability of the state like/V is computed as follows:

$$\begin{aligned}
 \text{PROB (like/V)} &= \text{MAX } (\text{PROB (flies/N)} * \text{PROB (V | N)}, \\
 &\quad \text{PROB (flies/V)} * \text{PROB (V | V)}) * \\
 &\quad \text{PROB (like/V)} \\
 &= \text{MAX } (.00725 * .43, 7.6 * 10^{-6} * .0001) * .1 = 3.12 \\
 &\quad * 10^{-4}
 \end{aligned}$$

The difference in this value from that shown in Figure 7.10 is simply a result of the fact that the calculation here used truncated approximate values for the probabilities. In other words, the most likely sequence of length two generating "Flies like" and ending in a V has a score of 3.1×10^{-4} (and is the sequence N V), the most likely one ending in a P has a score of 2.2×10^{-5} (and is the sequence N P), and the most likely one ending in an N has a score of 1.3×10^{-5} (and is the sequence N N). The heavier arrows indicate the best sequence leading up to each node. The computation continues in the same manner until each word has been processed. Figure 7.11 shows the result after the next iteration, and Figure 7.12 shows the final result. The highest probability sequence ends in

state flower/N. It is simple to trace back from this category (using BACKPTR(1, 4) and so on) to find the full sequence N V ART N, agreeing with our intuitions.

Algorithms like this can perform effectively if the probability estimates are computed from a large corpus of data that is of the same style as the input to be classified. Researchers consistently report labeling with 95 percent or better

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 204]

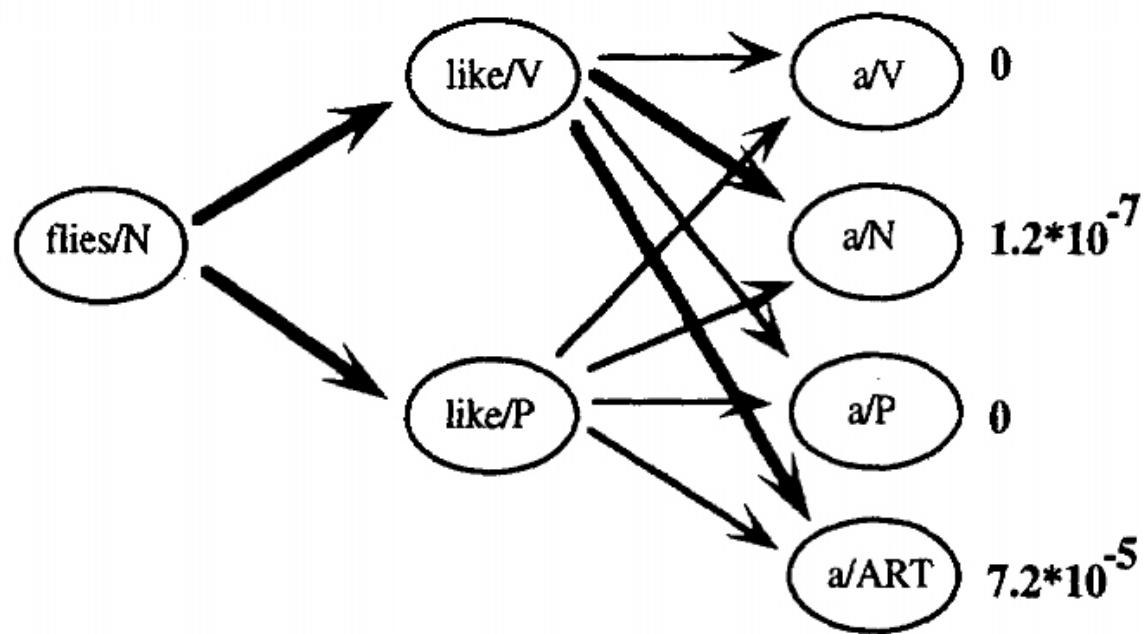


Figure 7.11 The result after the second iteration

Figure 7.11 The result after the second iteration

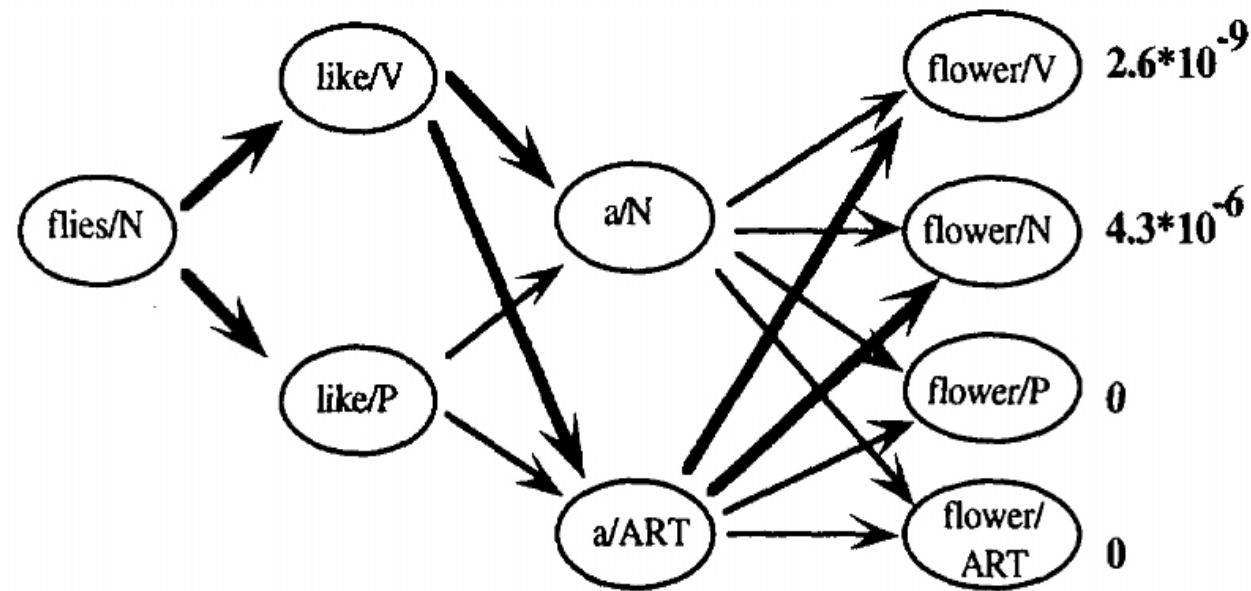


Figure 7.12 The result after the third iteration

Figure 7.12 The result after the third iteration

accuracy using trigram models. Remember, however, that the naive algorithm picks the most likely category about 90 percent of the time. Still, the error rate is cut in half by introducing these techniques.

>> [back](#)

7.4 Obtaining Lexical Probabilities

Corpus-based methods suggest some new ways to control parsers. If we had some large corpora of parsed sentences available, we could use statistical methods to identify the common structures in English and favor these in the parsing algorithm. This might allow us to choose the most likely interpretation when a sentence is ambiguous, and might lead to considerably more efficient parsers that are nearly deterministic. Such corpora of parsed sentences are now becoming available.

>> [back](#)

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 205]

BOX 7.1 Getting Reliable Statistics

Given that you need to estimate probabilities for lexical items and for n-grams, how much data is needed for these estimates to be reliable? In practice, the amount of data needed depends heavily on the size of the n-grams used, as the number of probabilities that need to be estimated grows rapidly with n. For example, a typical tagset has about 40 different lexical categories. To collect statistics on a unigram (a simple count of the words in each category), you would only need 40 statistics, one for each category. For bigrams, you would need 1600 statistics, one for each pair. For trigrams, you would need 64,000, one for each possible triple. Finally, for four-grams, you would need 2,560,000 statistics. As you can see, even if the corpus is a million words, a four-gram analysis would result in most categories being empty. For trigrams and a million-word corpus, however, there would be an average of 15 examples per category if they were evenly distributed. While the trigrams are definitely not uniformly distributed, this amount of data seems to give good results in practice.

One technique that is very useful for handling sparse data is called **smoothing**. Rather than simply using a trigram to estimate the probability of a category C_i , at position i, you use a formula that combines the trigram, bigram, and unigram statistics. Using this scheme, the probability of category C_i given the preceding categories C_1, \dots, C_{i-1} is estimated by the formula

$$\begin{aligned} \text{PROB } (C_1, \dots, C_{i-1}) \approx & \hat{U}_1 \text{ PROB } (C_i) + \hat{U}_2 \text{ PROB } (C_i \mid C_{i-1}) + \\ & \hat{U}_3 \text{ PROB } (C_i \mid C_{i-2}, C_{i-1}) \end{aligned}$$

where $\hat{U}_1 + \hat{U}_2 + \hat{U}_3 = 1$. Using this estimate, if the trigram has never been seen before, the bigram or unigram estimates still would guarantee a nonzero estimate in many cases where it is desired. Typically, the best performance will arise if X_3 is significantly greater than the other parameters so that the trigram information has the most effect on the probabilities. It is also possible to develop algorithms that learn good values for the parameters given a particular training set (for example, see Jelinek (1990)).

The first issue is what the input would be to such a parser. One simple approach would be to use a part-of-speech tagging algorithm from the last section to select a single category for each word and then start the parse with these categories. If the part-of-speech tagging is accurate, this will be an excellent approach, because a considerable amount of lexical ambiguity will be eliminated before the parser even starts. But if the tagging is wrong, it will prevent the parser from ever finding the correct interpretation. Worse, the parser may find a valid but implausible interpretation based on the wrongly tagged word and never realize the error. Consider that even at 95 percent accuracy, the chance that every word is correct in a sentence consisting of only 8 words is .67, and with 12 words it is .46 - less than half. Thus the chances of this approach working in general look slim.

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 206]

$PROB(ART the) \approx$.99	$PROB(N like) \approx$.16
$PROB(N flies) \approx$.48	$PROB(ART a) \approx$.995
$PROB(V flies) \approx$.52	$PROB(N a) \approx$.005
$PROB(V like) \approx$.49	$PROB(N flower) \approx$.78
$PROB(P like) \approx$.34	$PROB(V flower) \approx$.22

Figure 7.13 Context-independent estimates for the lexical categories

Figure 7.13 Context-independent estimates for the lexical categories

A more appropriate approach would be to compute the probability that each word appears in the possible lexical categories. If we could combine these probabilities with some method of assigning probabilities to rule use in the grammar, then we could develop a parsing algorithm that finds the most probable parse for a given sentence.

You already saw the simplest technique for estimating lexical probability by counting the number of times each word appears in the corpus in each category. Then the probability that word w appears in a lexical category L_j out

of possible categories L_1, \dots, L_N could be estimated by the formula

$$\text{PROB } (L_j | w) = \frac{\text{count}(L_j \text{ & } w)}{\sum_{i=1}^N \text{count}(L_i \text{ & } w)}$$

Using the data shown in Figure 7.5, we could derive the context-independent probabilities for each category and word shown in Figure 7.13.

As we saw earlier, however, such estimates are unreliable because they do not take context into account. A better estimate would be obtained by computing how likely it is that category L_i occurred at position t over all sequences given the input w_1, \dots, w_T . In other words, rather than searching for the one sequence that yields the maximum probability for the input, we want to compute the sum of the probabilities for the input from all sequences.

For example, the probability that "*flies*" is a noun in the sentence "*The flies like flowers*" would be calculated by summing the probability of all sequences that end with "*flies*" as a noun. Given the transition and lexical-generation probabilities in Figures 7.4 and 7.6, the sequences that have a nonzero values would be

The/ART flies/N $9.58 * 10^{-3}$

The/N flies/N $1.13 * 10^{-6}$

The/P flies/N $4.55 * 10^{-9}$

which adds up to $9.58 * 10^{-3}$. Likewise, three nonzero sequences end with "*flies*" as a V, yielding a total sum of $1.13 * 10^{-5}$. Since these are the only sequences that have nonzero scores when the second word is "*flies*", the sum of all these sequences will be the probability of the sequence "*The flies*", namely $9.591 * 10^{-3}$. We can now compute the probability that "*flies*" is a noun as follows:

```
PROB (flies/N | The flies)  
  
= PROB (flies/N & The flies) / PROB (The flies)
```

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 207]

Initialization Step

For $i = 1$ to N do

$$\text{SEQSUM}(i, 1) = \text{PROB}(w_1 | L_i) * \text{PROB}(L_i | \emptyset)$$

Computing the Forward Probabilities

For $t = 2$ to T do

 For $i = 1$ to N do

$$\text{SEQSUM}(i, t) = \sum_{j=1,N} (\text{PROB}(L_i | L_j) * \text{SEQSUM}(j, t-1)) * \text{PROB}(w_t | L_i)$$

Computing the Lexical Probabilities

For $t = 1$ to T do

 For $i = 1$ to N do

$$\text{PROB}(C_t = L_i) = \text{SEQSUM}(i, t) / \sum_{j=1,N} \text{SEQSUM}(j, t)$$

Figure 7.14 The forward algorithm for computing the lexical probabilities

Figure 7.14 The forward algorithm for computing the lexical probabilities

$$= 9.58 * 10^{-3} / 9.591 * 10^{-3}$$

$$= .9988$$

Likewise, the probability that "*flies*" is a verb would be .0012.

Of course, it would not be feasible to enumerate all possible sequences in a realistic example. Luckily, however, the same trick used in the Viterbi algorithm can be used here. Rather than selecting the maximum score for each node at each stage of the algorithm, we compute the sum of all scores.

To develop this more precisely, we define the forward probability, written as $\tilde{N}_i(t)$, which is the probability of producing the words w_1, \dots, w_t and ending in state w_t/L_i :

$$\tilde{N}_i(t) = \text{PROB } (w_t / L_i, w_1, \dots, w_t)$$

For example, with the sentence "*The flies like flowers*", $\tilde{N}_2(3)$ would be the sum of values computed for all sequences ending in V (the second category) in position 3 given the input "*The flies like*". Using the definition of

conditional probability, you can then derive the probability that word w_t is an instance of lexical category L_j as follows:

$$\text{PROB}(w_t / L_i, w_1, \dots, w_t) = \text{PROB}(w_t / L_i, w_1, \dots, w_t) / \\ \text{PROB}(w_1, \dots, w_t)$$

We estimate the value of $\text{PROB}(w_1, \dots, w_t)$ by summing over all possible sequences up to any state at position t , which is simply $\sum_{j=1, N} \tilde{N}_j(t)$. In other words, we end up with

$$\text{PROB}(w_t / L_i | w_1, \dots, w_t) \approx \sum_{j=1, N} \tilde{N}_j(t) / \sum_{j=1, N} \tilde{N}_j(t)$$

The first two parts of the algorithm shown in Figure 7.14 compute the forward probabilities using a variant of the Viterbi algorithm. The last step converts the

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 208]

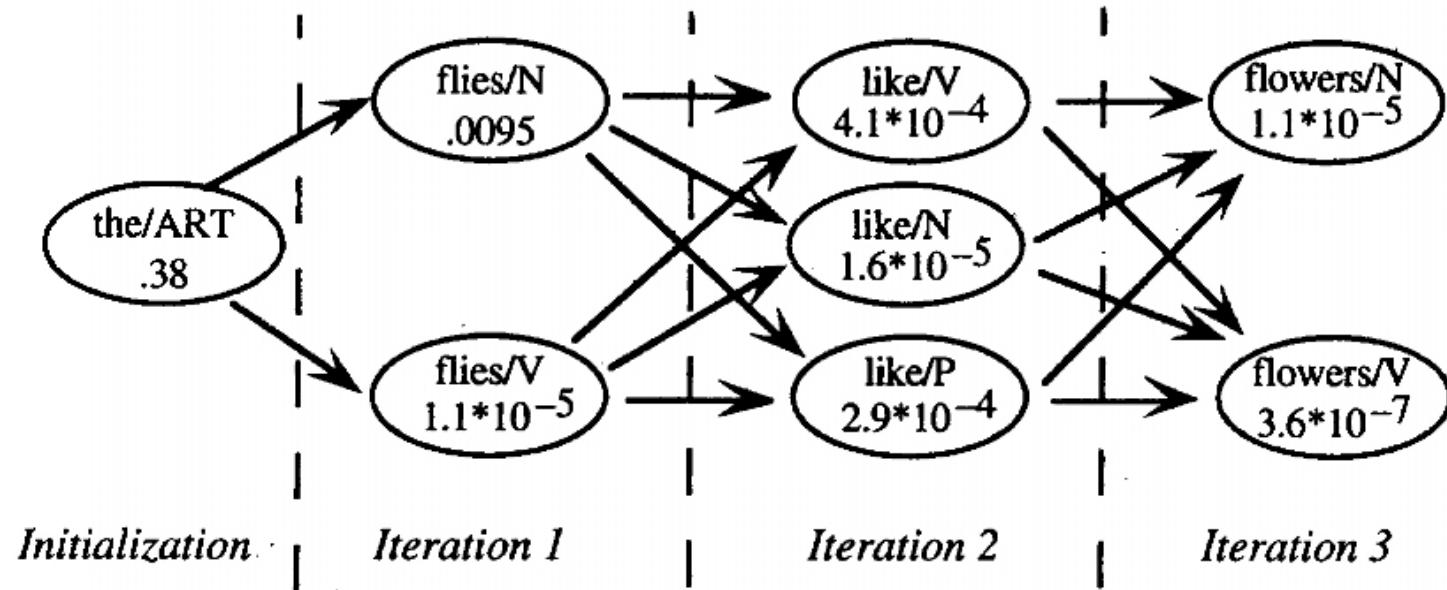


Figure 7.15 Computing the sums of the probabilities of the sequences

Figure 7.15 Computing the sums of the probabilities of the sequences

$PROB(\text{the/ART} \mid \text{the}) =$	1.0	$PROB(\text{like/P} \mid \text{the flies like}) \approx$	4
$PROB(\text{flies/N} \mid \text{the flies}) \approx$.9988	$PROB(\text{like/N} \mid \text{the flies like}) \approx$.022
$PROB(\text{flies/V} \mid \text{the flies}) \approx$.0011	$PROB(\text{flowers/N} \mid \text{the flies like flowers}) \approx$.967
$PROB(\text{like/V} \mid \text{the flies like}) \approx$.575	$PROB(\text{flowers/V} \mid \text{the flies like flowers}) \approx$.033

Figure 7.16 Context-dependent estimates for lexical categories in the sentence *The flies like flowers*

Figure 7.16 Context-dependent estimates for lexical categories in the sentence *The flies like flowers*

forward probabilities into lexical probabilities for the given sentence by normalizing the values.

Consider deriving the lexical probabilities for the sentence "*The flies like flowers*" using the probability estimates in Figures 7.4 and 7.6. The algorithm in Figure 7.14 would produce the sums shown in Figure 7.15 for each category in each position, resulting in the probability estimates shown in Figure 7.16.

Note that while the context-independent approximation in Figure 7.13 slightly favors the verb interpretation of "*flies*", the context-dependent approximation virtually eliminates it because the training corpus had no sentences

with a verb immediately following an article. These probabilities are significantly different than the context-independent ones and much more in line with intuition.

Note that you could also consider the **backward probability**, $\hat{O}_i(t)$, the probability of producing the sequence (w_1, \dots, w_t) beginning from state w_t/L_j . These values can be computed by an algorithm similar to the forward probability algorithm but starting at the end of the sentence and sweeping backward through the states. Thus a better method of estimating the lexical probabilities for word w_t would be to consider the entire sentence rather than just the words up to t . In this case, the estimate would be

$$\text{PROB}(w_t / L_i) = (\tilde{N}_i(t) * \hat{O}_i(t)) / \sum_{j=1, N} (\tilde{N}_i(t) * \hat{O}_j(t))$$

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 209]

Rule	Count for LHS	Count for Rule	Probability
1. $S \rightarrow NP\ VP$	300	300	1
2. $VP \rightarrow V$	300	116	.386
3. $VP \rightarrow V\ NP$	300	118	.393
4. $VP \rightarrow V\ NP\ PP$	300	66	.22
5. $NP \rightarrow NP\ PP$	1023	241	.24
6. $NP \rightarrow N\ N$	1023	92	.09
7. $NP \rightarrow N$	1023	141	.14
8. $NP \rightarrow ART\ N$	1023	558	.55
9. $PP \rightarrow P\ NP$	307	307	1

Grammar 7.17 A simple probabilistic grammar

Grammar 7.17 A simple probabilistic grammar

>> [back](#)

7.5 Probabilistic Context-Free Grammars

Just as finite state machines could be generalized to the probabilistic case, context-free grammars can also be generalized. To do this, we must have some statistics on rule use. The simplest approach is to count the number of times each rule is used in a corpus containing parsed sentences and use this to estimate the probability of each rule being used. For instance, consider a category C, where the grammar contains m rules, $R_1 R_m$, with the left-hand side C. You could estimate the probability of using rule R_j to derive C by the formula

$$\text{PROB}(R_j \mid C) \approx \frac{\text{Count}(\# \text{times } R_j \text{ used})}{\sum_{i=1}^m (\# \text{times } R_i \text{ used})}$$

Grammar 7.17 shows a probabilistic CFG with the probabilities derived from analyzing a parsed version of the demonstration corpus.

You can then develop algorithms similar in function to the Viterbi algorithm that, given a sentence, will find the most likely parse tree that could have generated that sentence. The technique involves making certain independence assumptions about rule use. In particular, you must assume that the probability of a constituent being derived by a rule R_j is independent of how the constituent is used as a subconstituent. For example, this

assumption would imply that the probabilities of NP rules are the same whether the NP is the subject, the object of a verb, or the object of a preposition. We know that this assumption is not valid in most cases. For instance, noun phrases in the subject position are much more likely to be pronouns than noun phrases not in the subject position. But, as before, it might be that useful predictive power can be obtained using these techniques in practice.

With this assumption, a formalism can be developed based on the probability that a constituent C generates a sequence of words w_i, w_{i+1}, \dots, w_j .

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 210]

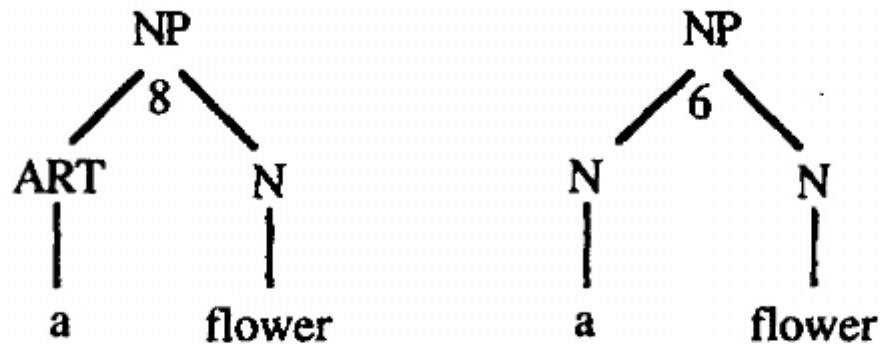


Figure 7.18 The two possible ways that *a flower* could be an NP

Figure 7.18 The two possible ways that *a flower* could be an NP

written as $w_{i,j}$. This type of probability is called the **inside probability** because it assigns a probability to the word sequence inside the constituent. It is written as

$$\text{PROB}(w_{i,j} \mid C)$$

Consider how to derive inside probabilities. The case for lexical categories is simple. In fact, these are exactly the lexical-generation probabilities derived in Section 7.3. For example, **PROB (flower | N)** is the inside probability that the constituent N is realized as the word "flower", which for our hypothetical corpus was .06, given in Figure 7.6.

Using such lexical-generation probabilities, we can then derive the probability that the constituent NP generates the sequence a "flower" as follows: There are only two NP rules in Grammar 7.17 that could generate a sequence of two words. The parse trees generated by these two rules are shown in Figure 7.18. You know the likelihood of each rule, estimated from the corpus as shown in Grammar 7.17, so the probability that the constituent NP generates the words a "flower" will be the sum of the probabilities of the two ways it can be derived, as follows:

$$\begin{aligned} \text{PROB}(a \text{ flower} | \text{NP}) &= \\ \text{PROB}(\text{Rule 8} | \text{NP}) * \text{PROB}(a | \text{ART}) * \text{PROB}(flower | \text{N}) + \\ \text{PROB}(\text{Rule 6} | \text{NP}) * \text{PROB}(a | \text{N}) * \text{PROB}(flower | \text{N}) \\ &= .55 * .36 * .06 + .09 * .001 * .06 \\ &= .012 \end{aligned}$$

This probability can then be used to compute the probability of larger constituents. For instance, the probability of generating the words "*A flower wilted*" from constituent S could be computed by summing the probabilities generated from each of the possible trees shown in Figure 7.19. Although there are three possible interpretations, the first two differ only in the derivation of a "*flower*" as an NP. Both these interpretations are already included in the preceding computation of **PROB (a flower | NP)**. Thus the probability of "*a flower blooms*" is

$$PROB(a \text{ flower } blooms \mid S) =$$

$$PROB(\text{Rule 1} \mid S) * PROB(a \text{ flower} \mid \text{NP}) * PROB(blooms \mid \text{VP}) +$$

$$PROB(\text{Rule 1} \mid S) * PROB(a \mid \text{NP}) * PROB(\text{flower } blooms \mid \text{VP})$$

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 211]

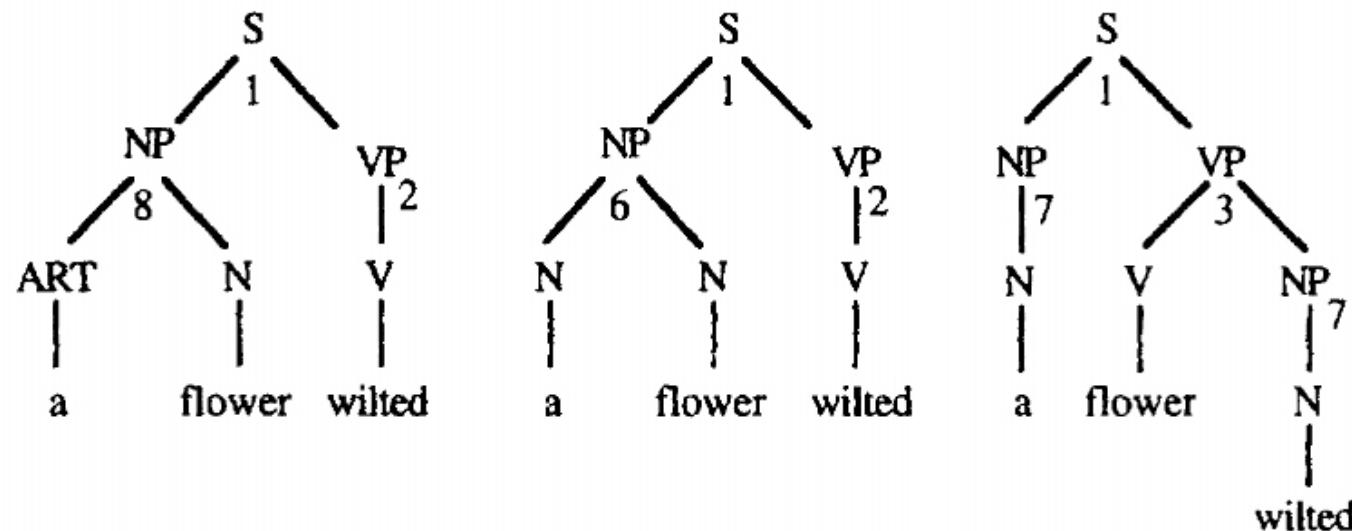


Figure 7.19 The three possible ways to generate *a flower wilted* as an S

Figure 7.19 The three possible ways to generate *a flower wilted* as an S

Using this method, the probability that a given sentence will be generated by the grammar can be computed efficiently. The method only requires recording the value of each constituent between each two possible positions.

In essence, it is using the same optimization that was gained by a packed chart structure, as discussed in Chapter 6.

We will not pursue this development further, however, because in parsing, you are interested in finding the most likely parse rather than the overall probability of a given sentence. In particular, it matters which of the two NP interpretations in Figure 7.19 was used. The probabilities of specific parse trees can be found using a standard chart parsing algorithm, where the probability of each constituent is computed from the probability of its subconstituents and the probability of the rule used. Specifically, when entering an entry E of category C using a rule i with n subconstituents corresponding to entries E_1, \dots, E_n , then

$$\text{PROB } (E) = \text{PROB } (\text{Rule } i \mid C) * \text{PROB } (E_1) * \dots * \text{PROB } (E_n)$$

For lexical categories it is better to use the forward probability than the lexical-generation probability. This will produce better estimates, because it accounts for some of the context of the sentence. You can use the standard chart parsing algorithm and add a step that computes the probability of each entry when it is added to the chart. Using the bottom-up algorithm and the probabilities derived from the demonstration corpus, Figure 7.20 shows the complete chart for the input "*a flower*". Note that the most intuitive reading, that it is an NP, has the highest probability by far, .54. But there are many other possible interpretations because of the low probability readings of the word *a* as a noun and the reading of "*flower*" as a verb. Note that the context-independent probabilities for "*flower*" would favor the verb interpretation, but the forward algorithm strongly favors the noun interpretation in the context where it immediately follows the word "*a*". The probabilities for the lexical constituents ART416, N417, V419, and N422 were computed using the forward algorithm. The context-independent lexical probabilities would be much less in line with intuition.

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 212]

	NP425	
	1 N422	.14
NP424		
1 N417		
2 N422		.00011
NP423		
1 ART416		
2 N422		.54
S421		
1 NP418		
2 VP420		3.2×10^{-8}
NP418	VP420	
1 N417 .00018		.00018
N417 .001	N422	.999
ART416 .99	V419	.00047

Figure 7.20 The full chart for a flower

Figure 7.20 The full chart for *a flower*

Unfortunately, a parser built using these techniques turns out not to work as well as you might expect, although it does help. Some researchers have found that these techniques identify the correct parse about 50 percent of the time. It doesn't do better because the independence assumptions that need to be made are too radical. Problems arise in many guises, but one critical issue is the handling of lexical items. For instance, the context-free model assumes that the probability of a particular verb being used in a VP rule is independent of which rule is being considered. This means lexical preferences for certain rules cannot be handled within the basic framework. This problem then influences many other issues, such as attachment decisions.

For example, Grammar 7.17 indicates that rule 3 is used 39 percent of the time, rule 4, 22 percent of the time, and rule 5, 24 percent of the time. This means that, independent of whatever words are used, an input sequence of form V NP PP will always be interpreted with the PP attached to the verb. The tree fragments are shown in Figure 7.21: Any structure that attaches the PP to the verb will have a probability of .22 from this fragment, whereas the structure that attaches the PP to the NP will have a probability of $.39 * .25$, namely .093. So the parser will always attach a PP to the verb phrase independent of what words are used. In fact, there are 23 cases of this situation in the corpus where the PP attaches to an NP, and the probabilistic parser gets every one of them wrong.

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 213]

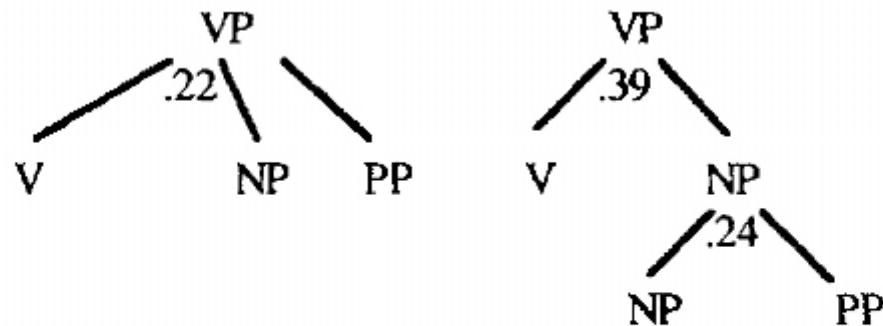


Figure 7.21 The two structures affecting attachment decisions

Figure 7.21 The two structures affecting attachment decisions

This is true even with particular verbs that rarely take a nppp complement. Because of the context-free assumption made about probabilities, the particular verb used has no effect on the probability of a particular rule being used. Because of this type of problem, a parser based on this approach with a realistic grammar is only a slight improvement over the nonprobabilistic methods.

For example, 84 sentences were selected from the corpus, each involving a PP attachment problem. With the standard bottom-up nonprobabilistic algorithm, using Grammar 7.17, the first complete S structure found was the correct answer on one-third of the sentences. By using probabilistic parsing, the highest probability S structure was correct half of the time. Thus, pure probabilistic parsing is better than guessing but leaves much to be desired. Note also that this test was on the same sentences that were used to compute the probabilities in the first place, so performance on new sentences not in the training corpus would tend to be even worse! Papers in the literature that use real corpora and extensive training report accuracy results similar to those described here.

While the pure probabilistic context-free grammars have their faults, the techniques developed here are important and can be reused in generalizations that attempt to develop more context-dependent probabilistic parsing schemes.

>> [back](#)

7.6 Best-First Parsing

So far, probabilistic context-free grammars have done nothing to improve the efficiency of the parser. Algorithms can be developed that attempt to explore the high-probability constituents first. These are called best-first parsing algorithms. The hope is that the best parse can be found quickly and much of the search space, containing lower-rated possibilities, is never explored.

It turns out that all the chart parsing algorithms in Chapter 3 can be modified fairly easily to consider the most likely constituents first. The central idea is to make the agenda a priority queue - a structure where the highest-rated elements are always first in the queue. The parser then operates by always removing the highest-ranked constituent from the agenda and adding it to the chart.

It might seem that this one change in search strategy is all that is needed to modify the algorithms, but there is a complication. The previous chart parsing algorithms all depended on the fact that the parser systematically worked from

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 214]

To add a constituent C from position p_1 to p_2 :

1. Insert C into the chart from position p_1 to p_2 .
2. For any active arc of the form $X \rightarrow X_1 \dots \circ C \dots X_n$ from position p_0 to p_1 , add a new active arc $X \rightarrow X_1 \dots C \circ \dots X_n$ from position p_0 to p_2 .

To add an active arc $X \rightarrow X_1 \dots C \circ C' \dots X_n$ to the chart from p_0 to p_2 :

1. If C is the last constituent (that is, the arc is completed), add a new constituent of type X to the agenda.
2. Otherwise, if there is a constituent Y of category C' in the chart from p_2 to p_3 , then recursively add an active arc $X \rightarrow X_1 \dots C C' \circ \dots X_n$ from p_0 to p_3 (which may of course add further arcs or create further constituents).

Figure 7.22 The new arc extension algorithm

Figure 7.22 The new arc extension algorithm

left to right, completely processing constituents occurring earlier in the sentence before considering later ones. With the modified algorithm, this is not the case. If the last word in the sentence has the highest score, it will be added to the chart first. The problem this causes is that you cannot simply add active arcs to the chart (and depend on later steps in the algorithm to extend them). In fact, the constituent needed to extend a particular active arc may already be on the chart. Thus, whenever an active arc is added to the chart, you must check to see if it can be extended immediately, given the current chart. Thus we need to modify the arc extension algorithm. The algorithm is as before (in Section 3.4), except that step 2 is modified to check for constituents that already exist on the chart. The complete algorithm is shown in Figure 7.22.

Adopting a best-first strategy makes a significant improvement in the efficiency of the parser. For instance, using Grammar 7.17 and lexicon trained from the corpus, the sentence "*The man put a bird in the house*" is parsed correctly after generating 65 constituents with the best-first parser. The standard bottom-up algorithm generates 158 constituents on the same sentence, only to obtain the same result. If the standard algorithm were modified to terminate when the first complete S interpretation is found, it would still generate 106 constituents for the same sentence. So best-first strategies can lead to significant improvements in efficiency.

Even though it does not consider every possible constituent, the best-first parser is guaranteed to find the highest probability interpretation. To see this, assume that the parser finds an interpretation S_1 with probability p_1 . The important property of the probabilistic scoring is that the probability of a constituent must always be lower (or equal) to the probability of any of its subconstituents. So if there were an interpretation S_2 that had a score p_2 , higher than p_1 , then it would have to be constructed out of subconstituents all with a probability of p_2 or higher. This means that all the subconstituents would have been added to the chart before S_1 was. But this means that the arc that constructs S_2 would be

>> [back](#)

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 215]

BOX 7.2 Handling Unknown Words

Another area where statistical methods show great promise is in handling unknown words. In traditional parsing, one unknown word will disrupt the entire parse. The techniques discussed in this chapter, however, suggest some interesting ideas. First, if you have a trigram model of the data, you may already have significant predictive power on the category of the unknown word. For example, consider a sequence $w_1 \ w_2 \ w_3$, where the last word is unknown. If the two previous words are in categories C_1 and C_2 , respectively, then you could pick the category C for the unknown word that maximizes $\text{PROB}(C \mid C_1 \ C_2)$. For instance, if C_2 is the category ART, then the most likely interpretation for C will probably be a noun (or maybe an adjective). Other techniques have been developed that use knowledge of morphology to predict the word

class. For instance, if the unknown word ends in *-ing* then it is likely to be a verb; if it ends in *-ly*, then it is likely to be an adverb; and so on. Estimates based on suffixes can be obtained by analyzing a corpus in the standard way to obtain estimates of **PROB (word is category C | end is -ly)** and so on. Techniques for handling unknown words are discussed in Church (1988), de Marcken (1990), and Weischedel et al. (1993).

completed, and hence S₂ would be on the agenda. Since S₂ has a higher score than S₁, it would be considered first.

While the idea of a best-first parser is conceptually simple, there are a few problems that arise when trying to apply this technique in practice. One problem is that if you use a multiplicative method to combine the scores, the scores for constituents tend to fall quickly as they cover more and more input. This might not seem problematic, but in practice, with large grammars, the probabilities drop off so quickly that the search closely resembles a breadth-first search: First build all constituents of length 1, then all constituents of length 2, and so on. Thus the promise of quickly finding the most preferred solution is not realized. To deal with this problem, some systems use a different function to compute the score for constituents. For instance, you could use the minimum score of any subconstituent and the rule used, that is,

$$\begin{aligned} \text{Score}(C) &= \text{MIN}(\text{Score}(C \rightarrow C_1, \dots, C_n), \text{Score}(C_1), \dots, \\ &\quad \text{Score}(C_n)) \end{aligned}$$

This gives a higher (or equal) score than the first approach, but a single poorly-rated subconstituent can essentially eliminate any constituent that contains it, no matter how well all the other constituents are rated. Unfortunately, testing the same 84 sentences using the MIN function leads to a significant decrease in accuracy to only 39 percent, not much better than a brute force search. But other researchers have suggested that the technique performs better than this in actual practice. You might also try other means of combining the scores, such as taking the average score of all subconstituents.

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 216]

Rule	the	house	peaches	flowers
$NP \rightarrow N$	0	0	.65	.76
$NP \rightarrow N\ N$	0	.82	0	0
$NP \rightarrow NP\ PP$.23	.18	.35	.24
$NP \rightarrow ART\ N$.76	0	0	0
Rule	ate	bloom	like	put
$VP \rightarrow V$.28	.84	0	.03
$VP \rightarrow V\ NP$.57	.1	.9	.03
$VP \rightarrow V\ NP\ PP$.14	.05	.1	.93

Figure 7.23 Some rule estimates based on the first word

Figure 7.23 Some rule estimates based on the first word

>> [back](#)

7.7 A Simple Context-Dependent Best-First Parser

The best-first algorithm leads to improvement in the efficiency of the parser but does not affect the accuracy problem. This section explores a simple alternative method of computing rule probabilities that uses more context-dependent lexical information. The idea exploits the observation that the first word in a constituent is often the head and thus has a dramatic effect on the probabilities of rules that account for its complement. This suggests a new probability measure for rules that is relative to the first word, $\text{PROB}(R \mid C, w)$. This is estimated as follows:

Count (# times rule R used for cat C starting with
w)

$\text{PROB}(R \mid C, w) = \frac{\text{Count}(\# \text{ times rule R used for cat C starting with } w)}{\text{Count}(\# \text{ times cat C starts with } w)}$

Count(# times cat C starts with w)

The effect of this modification is that probabilities are sensitive to the particular words used. For instance, in the corpus, singular nouns rarely occur alone as a noun phrase (that is, starting rule $\text{NP} \rightarrow \text{N}$), whereas plural nouns rarely are used as a noun modifying (that is, starting rule $\text{NP} \rightarrow \text{N N}$). This can be seen in the difference between the probabilities for these two rules given the words "*house*" and "*peaches*", shown in Figure 7.23. With the context-free probabilities, the rule $\text{NP} \rightarrow \text{N}$ had a probability of .14. This underestimates the rule when the input has a plural noun, and overestimates it when the input has a singular noun.

More importantly, the context-sensitive rules encode verb preferences for different subcategorizations. In particular, Figure 7.23 shows that the rule $\text{VP} \rightarrow \text{V NP PP}$ is used 93 percent of the time with the verb "*put*" but only 10 percent of the time with "*like*". This difference allows a parser based on these probabilities to do significantly better than the context-free probabilistic parser. In particular, on the same 84 test sentences on which the context-free probabilistic parser had 49 percent accuracy, the context-dependent probabilistic parser has 66 percent accuracy, getting the attachment correct on 14 sentences on which the context-

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 217]

Strategy	Accuracy on 84 PP Attachment Problems	Size of Chart Generated for <i>The man put the bird in the house</i>
Full Parse	33% (taking first S found)	158
Context-Free Probabilities	49%	65
Context-Dependent Probabilities	66%	36

Figure 7.24 A summary of the accuracy and efficiency of different parsing strategies

Figure 7.24 A summary of the accuracy and efficiency of different parsing strategies

VP6840					
1 V6812					
2 NP6815					
3 PP6822				.54	
VP6828					
1 V6812					
2 NP6827				.0038	
	NP6827				
	1 NP6815				
	2 PP6822			.13	
V6812	NP6815	PP6822			
.99	.76			.76	
put	the bird	in the house			

Figure 7.25 The chart for the VP *put the bird in the house*

Figure 7.25 The chart for the VP *put the bird in the house*

free parser failed. The context-dependent parser is also more efficient finding the answer on the sentence "*The man put the bird in the house*" generating only 36 constituents. The results of the three parsing strategies are summarized in Figure 7.24.

To see why the context-dependent parser does better, consider the attachment decision that has to be made between the trees shown in Figure 7.21 for the verbs "*like*" and "*put*", say in the sentences "*The man put the bird in the house*" and "*The man likes the bird in the house*". The context-free probabilistic parser would assign the same structure to both, getting the example with "*put*" right and the example with "*like*" wrong. The relevant parts of the charts are shown in Figures 7.25 and 7.26. In Figure 7.25 the probability of the rule **VP → V NP PP**, starting with "*put*", is .93, yielding a probability of .54 (.93 * .99 * .76 * .76) for constituent VP6840. This is far greater than the probability of .0038 given the alternative VP6828. Changing the verb to "*like*", as shown in Figure 7.26, affects the probabilities enough to override the initial bias towards the V-NP-PP interpretation. In this case, the probability of the rule **VP → V NP PP**, starting with "*like*", is only .1,

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 218]

VP6883					
1 V6848					.1
2 NP6873					
VP6881					
1 V6848					
2 NP6851					.054
3 PP6858					
	NP6873				
	1 NP6851				
	2 PP6858				.13
V6848	NP6851	PP6858			
.94	.76				.76
likes	the bird	in the house			

Figure 7.26 The chart for the VP *likes the bird in the house*

Figure 7.26 The chart for the VP *likes the bird in the house*

whereas the probability of rule **VP** → **V NP** is .9. This gives a probability of .1 to VP6883, beating out the alternative.

The question arises as to how accurate a parser can become by using the appropriate contextual information. While 66 percent accuracy was good compared to the alternative strategies, it still leaves a 33 percent error rate on sentences involving a PP attachment decision. What additional information could be added to improve the performance? Clearly, you could make the rule probabilities relative to a larger fragment of the input, using a bigram or trigram at the beginning of the rule. This may help the selectivity if there is enough training data.

Attachment decisions depend not only on the verb but also on the preposition in the PP. You could devise a more complex estimate based on the previous category, the head verb, and the preposition for rule **VP** → **V NP PP**. This would require more data to obtain reliable statistics but could lead to a significant increase in reliability. But a complication arises in that you cannot compare across rules as easily now: The **VP** → **V NP PP** rule is evaluated using the verb and the preposition, but there is no corresponding preposition with the rule **VP** → **V NP**. Thus a more complicated measure would have to be devised.

In general, the more selective the lexical categories, the more predictive the estimates can be, assuming there is enough data. Earlier we claimed that basing the statistics on words would require too much data. But is there a subset of words that could profitably be used individually? Certainly function words such as prepositions seem ideally suited for individual treatment. There is a fixed number of them, and they have a significant impact on the structure of the sentence. Similarly, other closed class words - articles, quantifiers, conjunctions, and so on - could all be treated individually rather than grouped together as a

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 219]

class. It is reasonable to assume we will be able to obtain enough data to obtain reliable estimates with these words.

The complications arise with the open class words. Verbs and nouns, for instance, play a major role in constraining the data, but there are far too many of them to consider individually. One approach is to model the common ones individually, as we did here. Another would be to try to cluster words together into classes based on their similarities. This could be done by hand, based on semantic properties. For instance, all verbs describing motion might have essentially the same behavior and be collapsed into a single class. The other option is to use some automatic technique for learning useful classes by analyzing corpora of sentences with attachment ambiguities.

>> [back](#)

Summary

Statistically based methods show great promise in addressing the ambiguity resolution problem in natural language processing. Such techniques have been used to good effect for part-of-speech tagging, for estimating lexical probabilities, and for building probabilistically based parsing algorithms. For part-of-speech tagging, the Viterbi algorithm using bigram or trigram probability models can attain accuracy rates of over 95 percent. Context-dependent lexical probabilities can be computed using an algorithm similar to the Viterbi algorithm that computes the forward probabilities. These can then be used as input to a chart parser that uses a context-free probabilistic grammar to find the most likely parse tree. The efficiency of parsers can be dramatically improved by using a best-first search strategy, in which the highest-rated constituents are added to the chart first. Unfortunately, the context-free probabilistic framework often does not identify the intuitively correct parse tree. Context-dependent probability measures can be developed that significantly improve the accuracy of the parser over the context-free methods. This is an area of active research, and significant developments can be expected in the near future.

>> [back](#)

Related Work and Further Readings

The literature on statistical methods of natural language processing is quite young, with the techniques being developed in the last decade. An excellent source that describes the techniques in detail is Charniak (1993). Many of the basic algorithms were developed in areas outside natural language processing. For instance, the Viterbi algorithm was originally developed on Markov models (Viterbi, 1967) and is in extensive use in a wide range of applications. The technique for developing probability estimates for lexical classes is based on the forward and forward-backward algorithms by Baum (1972). The use of these techniques for speech recognition is described well in an article by Rabiner (1989) and in many of the papers in Waibel and Lee (1990).

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 220]

The use of these techniques for part-of-speech tagging has been developed by many researchers, including Jelinek (1990), Church (1988), DeRose (1988), and Weischedel et al. (1993), to name a few. Many researchers have also developed parsers based on the context-independent rule probabilities described in Section 7.5, including Jelinek

(1990). One of the major attractions for statistically based approaches is the ability to learn effective parameters from processing corpora. These algorithms start with an initial estimate of the probabilities and then process the corpus to compute a better estimate. This can be repeated until further improvement is not found. These techniques are guaranteed to converge but do not necessarily find the optimal values. Rather, they find a local maximum and are thus similar to many hill-climbing algorithms. An example of a technique to learn parameters for a grammar from only partially analyzed data is given in Pereira and Schabes (1992).

Best-first parsing algorithms have been in use for a long time, going back to work such as that by Paxton and Robinson (1973). Only recently have they been explored using data derived from corpora (as in Weischedel et al. (1993) and Chitao and Gnshman (1990)). A good example of a context-dependent method using trigrams is that of Magerman and Weir (1992). In general, an excellent collection of recent work on statistical models can be found in a special issue of "*Computational Linguistics*" (Volume 19, Issues 1 and 2, 1993).

Prepositional phrase attachment seems to be a problem well suited to statistical techniques. Whittemore et al. (1990) explored the traditional strategies such as right association and minimal attachment and found that they did not have good predictive power, whereas lexical preference models looked promising. Hindle and Rooth (1993) explored some techniques for gathering such data from corpora that is not fully annotated with syntactic analyses.

The Brown corpus is described in Francis and Kucera (1982) and has been widely used as a testbed since its release. A good source for linguistic data is the Linguistic Data Consortium (LDC) housed at the University of Pennsylvania. The LDC has many large databases of language, in written and spoken forms, and in a wide range of styles and formats. It also contains the Penn Treebank data, which is a large corpus of sentences annotated with parses. The Penn Treebank is described in Marcus et al. (1993).

There are many introductory books on probability theory available. One good one, by Ross (1988), introduces the basic concepts and includes brief discussions of Markov chains and information theory.

>> [back](#)

Exercises for Chapter 7

1. (easy) Prove Bayes' rule by using the definition of conditional probability. Also prove that if A and B are independent then $\text{PROB } (A \mid B) = \text{PROB } (A)$.
2. (medium) Say we have an acceptable margin of error of between .4 and .6 for estimating the probability of a fair coin coming up heads. What is the chance of obtaining a reliable estimate for the probability with five flips?

[Allen 1995: Chapter 7 - Ambiguity Resolution: Statistical Methods 221]

3. (medium) Hand-simulate the Viterbi algorithm using the data and probability estimates in Figures 7.4 - 7.6 on the sentence "*Flower flowers like flowers*". Draw transition networks as in Figures 7.10 - 7.12 for the problem, and identify what part of speech the algorithm identifies for each word.

4. (medium) Using the bigram and lexical-generation probabilities given in this chapter, calculate the word probabilities using the forward algorithm for the sentence "*The a flies like flowers*" (involving a very rare use of the word a as a noun, as in "*the a flies*", "*the b flies*", and so on). Remember to use .0001 as a probability for any bigram not in the table. Are the results you get reasonable? If not, what is the problem and how might it be fixed?"

5. (medium) Using the tagged corpus provided, write code to collect bigram and lexical statistics, and implement a part-of-speech tagger using the Viterbi algorithm. Test your algorithm on the same corpus. How many sentences are tagged correctly? How many category errors did this involve? Would you expect to get better accuracy results on the sentences that involved an error if you had access to more data?

6. (medium) Consider an extended version of Grammar 7.17 with the additional rule

10. **VP** → **V PP**

The revised rule probabilities are shown here. (Any not mentioned are the same as in Grammar 7.17.)

VP → **V** .32

VP → **V NP** .33

VP → **V NP PP** .20

VP → **V PP** .15

In addition, the following bigram probabilities differ from those in Figure 7.4:

$$\text{PROB (N | V)} = .53$$

$$\text{PROB (ART | V)} = .32$$

$$\text{PROB (P | V)} = .15$$

- a. Hand-simulate (or implement) the forward algorithm on *Fruit flies like birds* to produce the lexical probabilities.
- b. Draw out the full chart for "*Fruit flies like birds*", showing the probabilities of each constituent.

7. (hard) Implement an algorithm to derive the forward and backward probabilities for lexical categories. Develop two lexical analysis programs: one that uses only the forward algorithm, and one that combines both techniques. Give a sentence for which the results differ substantially.

8. (*hard*) Generalize the Viterbi algorithm to operate based on trigram statistics. Since the probability of a state at position t now depends on the two preceding states, this problem does not fit the basic definition of the Markov assumption. It is often called a second-order Markov assumption. The best way to view this is as an HMM in which all the states are labeled by a pair of categories. Thus, the sequence **ART N V N** could be generated from a sequence of states labeled **(Ȳ, ART)**, **(ART, N)**, **(N, V)**, and **(V, N)**. The transition probabilities between these states would be the trigram probabilities. For example, the probability of the transition from state **(Ȳ, ART)** to **(ART, N)** would be the probability of category N following the categories **Ȳ** and **ART**. Train your data using the supplied corpus and implement a part-of-speech tagger based on trigrams. Compare its performance on the same data with the performance of the bigram model.

>> [back](#)

Allen 1995 : Natural Language Understanding

	<u>Contents</u>	<u>Preface</u>	<u>Introduction</u>	
<u>previous chapter</u>	<u>Part I Syntactic Processing</u>	<u>Part II Semantic Interpretation</u>	<u>Part III - Context / World Knowledge</u>	next chapter
	<u>Appendices</u>	<u>Bibliography</u>		
	<u>Summaries</u>	<u>Further Readings</u>	<u>Exercises</u>	

Chapter 8 : Semantics and Logical Form

[8.1 Semantics and Logical Form](#)

[8.2 Word Senses and Ambiguity](#)

[8.3 The Basic Logical Form Language](#)

[8.4 Encoding Ambiguity in the Logical Form](#)

[8.5 Verbs and States in Logical Form](#)

[8.6 Thematic Roles](#)

[8.7 Speech Acts and Embedded Sentences](#)

[8.8 Defining Semantic Structure: Model Theory](#)

[Summary](#)

8.1 Semantics and Logical Form

Related Work and Further Readings

Exercises for Chapter 8

[Allen 1995: Chapter 8 – Semantics and Logical Form / 227]

This chapter introduces the basic ideas underlying theories of meaning, or semantics. It introduces a level of context-independent meaning called the logical form, which can be produced directly from the syntactic structure of a sentence. Because it must be context independent, the logical form does not contain the results of any analysis that requires interpretation of the sentence in context.

Section 8.1 introduces the basic notions of meaning and semantics, and describes the role of a logical form in semantic processing. Section 8.2 introduces word senses and the semantic primitives, and discusses the problem of word-sense ambiguity. Section 8.3 then develops the basic logical form language for expressing the context-independent meaning of sentences. This discussion is extended in Section 8.4 with constructs that concisely encode certain common forms of ambiguity. Section 8.5 discusses the representation of verbs and introduces the notion of state and event variables. Section 8.6 then discusses thematic roles, or cases, and shows how they can be used to capture various semantic generalities across verb meanings. Section 8.7 introduces the notion of surface speech acts and discusses the treatment of embedded sentences in the logical form. This completes the description of the logical form language, which is concisely characterized in Figures 8.7 and 8.8. The optional Section 8.8 is for the student interested in defining a model-theoretic semantics for the logical form language itself. It describes a model theory for the logical form language and discusses various semantic relationships among sentences that can be defined in terms of entailment and implicature.

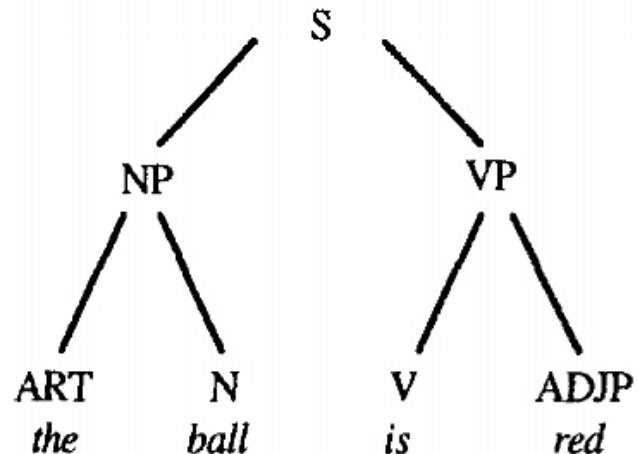
>> [back](#)

8.1 Semantics and Logical Form

Precisely defining the notions of semantics and meaning is surprisingly difficult because the terms are used for several different purposes in natural and technical usage. For instance, there is a use of the verb "*mean*" that has nothing to do with language. Say you are walking in the woods and come across a campfire that is just noticeably warm. You might say "*This fire means someone camped here last night*". By this you mean that the fire is evidence for the conclusion or implies the conclusion. This is related to, but different from, the notion of meaning that will be the focus of the next few chapters. The meaning we want is closer to the usage when defining a word, such as in the sentence "*'Amble' means to walk slowly*". This defines the meaning of a word in terms of other words. To make this more precise, we will have to develop a more formally specified language in which we can specify meaning without having to refer back to natural language itself. But even if we can do this, defining a notion of sentence meaning is difficult. For example, I was at an airport recently and while I was walking towards my departure gate, a guard at the entrance asked, "Do you know what gate you are going to?" I interpreted this as asking whether I knew where I was going and answered yes. But this response was based on a misunderstanding of what the guard meant, as he then asked, "Which gate is it?" He clearly had wanted me to

[Allen 1995: Chapter 8 – Semantics and Logical Form / 228]

SYNTACTIC
ANALYSIS



semantic interpretation

LOGICAL
FORM

(RED1 <THE b1 BALL>)

contextual interpretation

FINAL REPRESENTATION

Red(BO73)

Figure 8.1 Logical form as an intermediate representation

Figure 8.1 Logical form as an intermediate representation

tell him the gate number. Thus the sentence "*Do you know what gate you are going to?*" appears to mean different things in different contexts.

Can we define a notion of sentence meaning that is independent of context? In other words, is there a level at which the sentence "*Do you know what gate you are going to?*" has a single meaning, but may be used for different purposes? This is a complex issue, but there are many advantages to trying to make such an approach work. The primary argument is modularity. If such a division can be made, then we can study sentence meaning in detail without all the complications of sentence usage. In particular, if sentences have no context-independent meaning, then we may not be able to separate the study of language from the study of general human reasoning and context. As you will see in the next few chapters, there are many examples of constraints based on the meaning of words that appear to be independent of context. So from now on, we will use the term "*meaning*" in this context-independent sense, and we will use the term "*usage*" for the context-dependent aspects. The representation of context-independent meaning is called the logical form. The process of mapping a sentence to its logical form is called semantic interpretation, and the process of mapping the logical form to the final knowledge representation (KR) language is called contextual interpretation. Figure 8.1 shows a simple version of the stages of interpretation. The exact meaning of the notations used will be defined later.

For the moment let us assume the knowledge representation language is the first-order predicate calculus (FOPC). Given that assumption, what is the status

[Allen 1995: Chapter 8 – Semantics and Logical Form / 229]

of the logical form? In some approaches the logical form is defined as the literal meaning of the utterance, and the logical form language is the same as the final knowledge representation language. If this is to be a viable approach in the long run, however, it would mean that the knowledge representation must be considerably more complex than representations in present use in AI systems. For instance, the logical form language must allow indexical terms, that is, terms that are defined by context. The pronouns "*I*" and "*you*" are indexical because their interpretation depends on the context of who is speaking and listening. In fact most definite descriptions (such as "*the red ball*") are indexical, as the object referred to can only be identified with respect to a context. Many other aspects of language, including the interpretation of tense and determining the scope of quantifiers, depend on context as well and thus cannot be uniquely determined at the logical form level. Of course, all of this could be treated as ambiguity at the logical form level, but this would be impractical, as every sentence would have large numbers of possible logical forms (as in the sentence "*The red ball dropped*", which would have a different logical form for every possible object that could be described as a ball that is red).

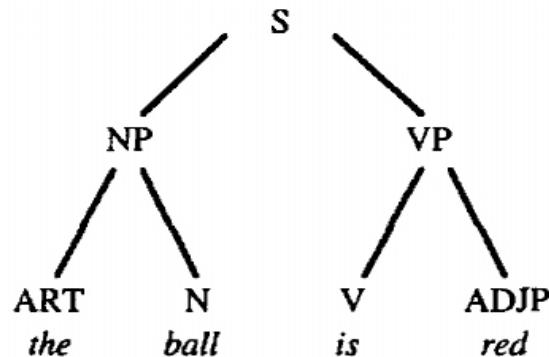
But if the logical form language is not part of the knowledge representation language, what is its formal status? A promising approach has been developed in linguistics over the last decade that suggests an answer that uses the

notion of a situation, which is a particular set of circumstances in the world. This corresponds reasonably well to the intuitive notion of the meaning of "*situation*" in English. For instance, when attending a class, you are in a situation where there are fellow students and an instructor, where certain utterances are made by the lecturer, questions asked, and so on. Also, there will be objects in the lecture hall, say a blackboard and chairs, and so on. More formally, you might think of a situation as a set of objects and relations between those objects. A very simple situation might consist of two objects, a ball B0005 and a person P86, and include the relationship that the person owns the ball. Let us encode this situation as the set $\{(BALL\ B0005), (PERSON\ P86), (OWNS\ P86\ B0005)\}$.

Language creates special types of situations based on what information is conveyed. These issues will be explored in detail later, but for now consider the following to help your intuition. In any conversation or text, assume there is a discourse situation that records the information conveyed so far. A new sentence is interpreted with respect to this situation and produces a new situation that includes the information conveyed by the new sentence. Given this view, the logical form is a function that maps the discourse situation in which the utterance was made to a new discourse situation that results from the occurrence of the utterance. For example, assume that the situation we just encoded has been created by some preceding sentences describing the ball and who owns it. The utterance "*The ball is red*" might produce a new situation that consists of the old situation plus the new fact that B0005 has the property RED: $((BALL\ B0005), (PERSON\ P86), (OWNS\ P86\ B0005), (RED\ B0005))$. Figure 8.2 shows this view of the interpretation process, treating the logical form as a function between

[Allen 1995: Chapter 8 – Semantics and Logical Form / 230]

SYNTACTIC
ANALYSIS



semantic interpretation

LOGICAL
FORM

(ASSERT (RED1 <THE b1 BALL>))

{(BALL B0005),
(PERSON P86),
(OWNS P86 B0005)}

CONTEXTUAL
INTERPRETATION

{(BALL B0005),
(PERSON P86),
(OWNS P86 B0005),
(RED B0005)}

INITIAL DISCOURSE
SITUATION

UPDATED DISCOURSE
SITUATION

Figure 8.2 Logical form as a function

Figure 8.2 Logical form as a function

situations. The two organizations presented in Figures 8.1 and 8.2 differ in that the latter might not include a single identifiable expression in the knowledge representation that fully captures the "meaning" of the sentence. Rather, the logical form might make a variety of changes to produce the updated situation. This allows other implications to be derived from an utterance that are not directly captured in the semantic content of the sentence. Such issues will become important later when we discuss contextual interpretation.

Even though much of language is highly context dependent, there is still considerable semantic structure to language that is context independent and that can be used in the semantic interpretation process to produce the logical form. Much of this semantic knowledge consists of the type of information you can find in dictionaries - the basic semantic properties of words (that is, whether they refer to relations, objects, and so on), what different senses are possible for each word, what senses may combine to form larger semantic structures, and so on. Identifying these forms of information and using this information to compute a logical form are the focus of Part II of the book.

>> [back](#)

[Allen 1995: Chapter 8 – Semantics and Logical Form / 231]

8.2 Word Senses and Ambiguity

To develop a theory of semantics and semantic interpretation, we need to develop a structural model, just as we did for syntax. With syntax we first introduced the notion of the basic syntactic classes and then developed ways to constrain how simple classes combine to form larger structures. We will follow the same basic strategy for semantics. You might think that the basic semantic unit could be the word or the morpheme, but that approach runs into problems because of the presence of ambiguity. For example, it is not unusual for the verb *go* to have more than 40 entries in a typical dictionary. Each one of these definitions reflects a different sense of the word. Dictionaries often give synonyms for particular word senses. For *go* you might find synonyms such as *move*, *depart*, *pass*, *vanish*, *reach*, *extend*, and *set out*. Many of these highlight a different sense of the verb *go*. Of course, if these are true synonyms of some sense of *go*, then the verbs themselves will share identical senses. For instance, one of the senses of *go* will be identical to one of the senses of *depart*.

If every word has one or more senses then you are looking at a very large number of senses, even given that some words have synonymous senses. Fortunately, the different senses can be organized into a set of broad classes of objects by which we classify the world. The set of different classes of objects in a representation is called its ontology. To handle a natural language, we need a much broader ontology than commonly found in work on formal logic. Such classifications of objects have been of interest for a very long time and arise in the writings of Aristotle (384—322 B.C.). The major classes that Aristotle suggested were substance (physical objects), quantity (such as numbers), quality (such as bright red), relation, place, time, position, state, action, and affection. To this list we might add other classes such as events, ideas, concepts, and plans. Two of the most influential classes are

actions and events. Events are things that happen in the world and are important in many semantic theories because they provide a structure for organizing the interpretation of sentences. Actions are things that agents do, thus causing some event. Like all objects in the ontology, actions and events can be referred to by pronouns, as in the discourse fragment

We lifted the box. It was hard work.

Here, the pronoun "it" refers to the action of lifting the box. Another very influential category is the situation. As previously mentioned, a situation refers to some particular set of circumstances and can be viewed as subsuming the notion of events. In many cases a situation may act like an abstraction of the world over some location and time. For example, the sentence "*We laughed and sang at the football game*" describes a set of activities performed at a particular time and location, described as the situation "*the football game*".

Not surprisingly, ambiguity is a serious problem during semantic interpretation. We can define a word as being semantically ambiguous if it maps to more than one sense. But this is more complex than it might first seem, because we need to have a way to determine what the allowable senses are. For example,

[Allen 1995: Chapter 8 – Semantics and Logical Form / 232]

intuitively the word "*kid*" seems to be ambiguous between a baby goat and a human child. But how do we know it doesn't have a single sense that includes both interpretations? The word "*horse*", on the other hand, seems not to be ambiguous even though we know that horses may be subdivided into mares, colts, trotters, and so on. Why doesn't the word "*horse*" seem ambiguous when each time the word is used we might not be able to tell if it refers to a mare or a colt? A few linguistic tests have been suggested to define the notion of semantic ambiguity more precisely. One effective test exploits the property that certain syntactic constructs typically require references to identical classes of objects. For example, the sentence "*I have two kids and George has three*" could mean that George and I are goat farmers or that we have children, but it can't mean a combination of both (I have goats and George has children). On the other hand you can say "*I have one horse and George has two*", even when I have a colt and George has mares. Thus this test provides a way to examine our intuitions about word senses. The word "*kid*" is ambiguous between two senses, BABY-GOAT1 and BABY-HUMAN1, whereas "*horse*" is not ambiguous between MARE1 and COLT1 but rather has a sense HORSE1 that includes both. This brings up the important point that some senses are more specific than others. This property is often referred to as vagueness. The sense HORSE1 is vague to the extent that it doesn't distinguish between mares and colts. Of course, the sense MARE1 is vague with respect to whether it is a large mare or a small one. Virtually all senses involve some degree of vagueness, as they might always allow some more precise specification.

A similar ambiguity test can be constructed for verb senses as well. For example, the sentence "*I ran last year and George did too*" could mean that we both were candidates in an election or that we both ran some race, but it would be difficult to read it as a mixture of the two. Thus the word *run* is ambiguous between the senses RUN1 (the exercise sense) and RUN2 (the political sense). In contrast, the verb "*kiss*" is vague in that it does not specify where one is kissed. You can say "*I kissed Sue and George did too*", even though I kissed her on the cheek and George kissed her hand.

In addition to lexical ambiguity, there is considerable structural ambiguity at the semantic level. Some forms of ambiguity are parasitic on the underlying syntactic ambiguity. For instance, the sentence "*Happy cats and dogs live on the farm*" is ambiguous between whether the dogs are also happy or not (that is, is it happy cats and happy dogs, or happy cats and dogs of any disposition). Although this ambiguity does have semantic consequences, it is actually rooted in the syntactic structure; that is, whether the conjunction involves two noun phrases, (Happy cats) and (dogs), or the single noun phrase (Happy (cats and dogs)). But other forms of structural ambiguity are truly semantic and arise from a single syntactic structure. A very common example involves quantifier scoping. For instance, does the sentence "*Every boy loves a dog*" mean that there is a single dog that all boys love, or that each boy might love a different dog? The syntactic structure is the same in each case, but the difference lies in how the quantifiers

[Allen 1995: Chapter 8 – Semantics and Logical Form / 233]

are scoped. For example, the two meanings correspond roughly to the following statements in FOPC:

$\exists d. \text{Dog}(d) \& \forall b. \text{Boy}(b) \rightarrow \text{Loves}(b,d)$

$\forall b. \text{Boy}(b) \rightarrow \exists d. \text{Dog}(d) \& \text{Loves}(b,d)$

Thus, while *Every boy loves a dog* has a single syntactic structure, its semantic structure is ambiguous. Quantifiers also vary with respect to vagueness. The quantifier *all* is precise in specifying every member of some set, but a quantifier such as *many*, as in *Many people saw the accident*, is vague as to how many people were involved.

You might also think that indexical terms such as *you*, *I*, and *here* are ambiguous because their interpretations depend on context. But this is not the sense of ambiguity being discussed here. Note that the word *dog* is not considered ambiguous because there are many dogs to which the noun phrase *the dog* could refer. The semantic meaning of *the dog* is precisely determined. Likewise, the pronoun *I* may be unspecified as to its referent, but it has a single well-defined sense.

While the referents of phrases are context dependent and thus beyond the scope of this discussion, there are context-independent constraints on reference that must be accounted for. Consider the sentence *Jack saw him in the mirror*. While it is not specified who was seen, you know that it wasn't Jack seeing himself. To express this meaning, you would have to use the reflexive pronoun, as in *Jack saw himself in the mirror*. Thus certain reference constraints arise because of the structure of sentences. This topic will be explored in Chapter 12.

A very important aspect of context-independent meaning is the cooccurrence constraints that arise between word senses. Often the correct word sense can be identified because of the structure and meaning of the rest of the sentence. Consider the verb "*run*", which has one sense referring to the action you do when jogging and another referring to the action of operating some machine. The first sense is typically realized as an intransitive verb (as in *Jack ran in the park*), whereas the second can only be realized as a transitive verb (as in *Jack ran the printing press for years*). In other cases the syntactic structure remains the same, but the possible senses of the words can

only combine in certain ways. For instance, consider *Jack ran in the park* versus *Jack ran in the election*. The syntactic structures of these sentences are identical, but different senses of the verb *run* must be selected because of the possible senses in the modifier. One of the most important tasks of semantic interpretation is to utilize constraints such as this to help reduce the number of possible senses for each word.

>> [back](#)

8.3 The Basic Logical Form Language

The last section introduced a primitive unit of meaning, namely the word sense. This section defines a language in which you can combine these elements to form meanings for more complex expressions. This language will resemble FOPC,

[Allen 1995: Chapter 8 – Semantics and Logical Form / 234]

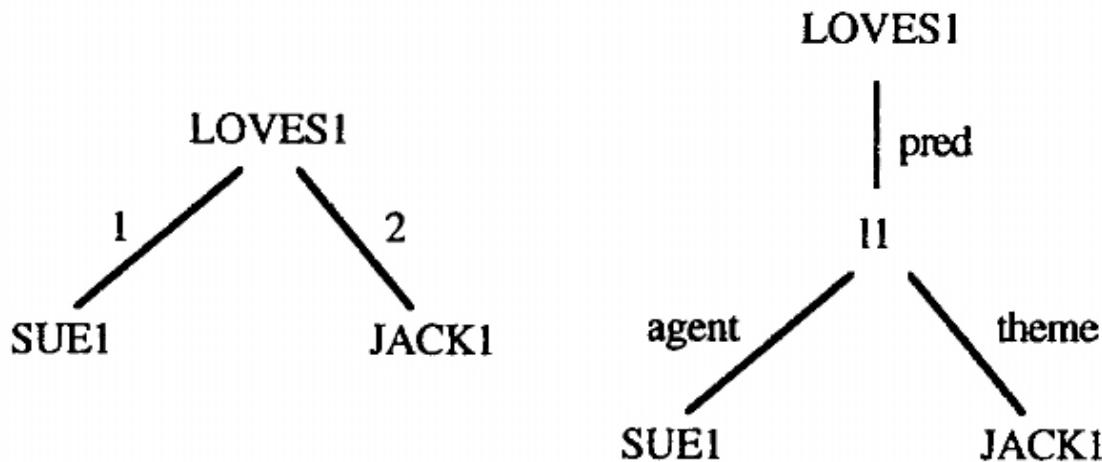


Figure 8.3 Two possible network representations of *Sue loves Jack*

Figure 8.3 Two possible network representations of "Sue loves Jack"

although there are many equivalent forms of representation, such as network-based representations, that use the same basic ideas. The word senses will serve as the atoms or constants of the representation. These constants can be classified by the types of things they describe. For instance, constants that describe objects in the world,

including abstract objects such as events and situations, are called terms. Constants that describe relations and properties are called predicates. A proposition in the language is formed from a predicate followed by an appropriate number of terms to serve as its arguments. For instance, the proposition corresponding to the sentence "*Fido is a dog*" would be constructed from the term FIDO1 and the predicate constant DOG1 and is written as

(DOG1 FIDO1)

Predicates that take a single argument are called **unary predicates** or **properties**; those that take two arguments, such as LOVES1, are called **binary predicates**; and those that take n arguments are called **n -ary predicates**. The proposition corresponding to the sentence *Sue loves Jack* would involve a binary predicate LOVES1 and would be written as

(LOVES1 SUE1 JACK1)

You can see that different word classes in English correspond to different types of constants in the logical form. Proper names, such as *Jack*, have word senses that are terms; common nouns, such as *dog*, have word senses that are unary predicates; and verbs, such as *run*, *love*, and *put*, have word senses that correspond to n -ary predicates, where n depends on how many terms the verb subcategorizes for.

Note that while the logical forms are presented in a predicate-argument form here, the same distinctions are made in most other meaning representations. For instance, a network representation would have nodes that correspond to the word senses and arcs that indicate the predicate-argument structure. The meaning of the sentence *Sue loves Jack* in a semantic networklike representation might appear in one of the two forms shown in Figure 8.3. For most purposes all of these representation formalisms are equivalent.

[Allen 1995: Chapter 8 – Semantics and Logical Form / 235]

More complex propositions are constructed using a new class of constants called logical operators. For example, the operator NOT allows you to construct a proposition that says that some proposition is not true. The proposition corresponding to the sentence *Sue does not love Jack* would be

(NOT (LOVES1 SUE1 JACK1))

English also contains operators that combine two or more propositions to form a complex proposition. FOPC contains operators such as disjunction (v), conjunction (&), what is often called implication (c), and other forms (there are 16 possible truth functional binary operators in FOPC). English contains many similar operators including *or*, *and*, *if only if*, and so on. Natural language connectives often involve more complex relationships between sentences. For instance, the conjunction "and" might correspond to the logical operator "&" but often also involves temporal sequencing, as in "*I went home and had a drink*", in which going home preceded having the drink. The connective "but", on the other hand, is like "and" except that the second argument is something that the hearer might not expect to be true given the first argument. The general form for such a proposition is (*connective proposition proposition*). For example, the logical form of the sentence "*Jack loves Sue or Jack loves Mary*" would be (OR1 (LOVES1 JACK1 SUE1) (LOVES1 JACK1 MARY1)). The logical form language will allow both operators corresponding to word senses and operators like "&" directly from FOPC. The logic-based operators will be used to connect propositions not explicitly conjoined in the sentence.

With the simple propositions we just defined, we can define the meaning of only a very limited subset of English, namely sentences consisting of simple verb forms and proper names. To account for more complex sentences, we must define additional semantic constructs. One important construct is the quantifier. In first-order predicate calculus, there are only two quantifiers: \exists and \forall . English contains a much larger range of quantifiers, including *all*, *some*, *most*, *many*, *a few*, *the*, and so on. To allow quantifiers, variables are introduced as in first-order logic but with an important difference. In first-order logic a variable only retains its significance within the scope of the quantifier. Thus two instances of the same variable x occurring in two different formulas - say in the formulas \mathbf{jx} , $\mathbf{p(x)}$ and \mathbf{jx} , $\mathbf{Q(x)}$ - are treated as completely different variables with no relation to each other. Natural languages display a different behavior. For instance, consider the two sentences "*A man entered the room. He walked over to the table.*" The first sentence introduces a new object to the discussion, namely some man. You might think to treat the meaning of this sentence along the lines of the existential quantifier in logic. But the problem is that the man introduced existentially in the first sentence is referred to by the pronoun "*He*" in the second sentence. So variables appear to continue their existence after being introduced. To allow this, each time a discourse variable is introduced, it is given a unique name not used before. Under the right circumstances, a subsequent sentence can then refer back to this term.

[Allen 1995: Chapter 8 – Semantics and Logical Form / 236]

Quantifier	Use	Example
THE	definite reference	the dog
A	indefinite reference	a dog
BARE	bare singular NP (mass term) or	water, food
BARE	bare plural NP (generics)	dogs

Figure 8.4 Some common quantifiers

Figure 8.4 Some common quantifiers

Natural language quantifiers have restricted ranges and thus are more complex than those found in FOPC. In FOPC a formula of the form $\exists_x. P_x$ is true if and only if P_x is true for every possible object in the domain (that is, x may be any term in the language). Such statements are rare in natural language. Rather, you would say "*all dogs bark*" or "*most people laughed*", which require constructs that are often called generalized quantifiers. These quantifiers are used in statements of the general form

(quantifier variable : restriction-proposition body-proposition)

For instance, the sentence "*Most dogs bark*" would have the logical form

(**MOST1** d1 (**DOG1** d1) (**BARKS1** d1))

This means that most of the objects d1 that satisfy (**DOG1** d1) also satisfy (**BARKS1** d1). Note that this has a very different meaning from the formula

(**MOST** d2 : (**BARKS1** d2) (**DOG1** d2))

which roughly captures the meaning of the sentence "*Most barking things are dogs*".

A very important class of generalized quantifiers corresponds to the articles "*the*" and "*a*". The sentence "*The dog barks*" would have a logical form

(**THE** x: (**DOG1** x) (**BARKS1** x))

which would be true only if there is a uniquely determined dog in context and that dog barks. Clearly, in any natural setting there will be many dogs in the world, so the use of context to identify the correct one is crucial for understanding the sentence. Since this identification process requires context, however, discussion of it is delayed until Part III. Here it suffices to have a way to write the logical form. The special set of quantifiers corresponding to the articles (or the absence of articles in bare noun phrases) is shown in Figure 8.4.

More complex noun phrases will result in more complex restrictions. For instance, the sentence "*The happy dog barks*" will involve a restriction that is a conjunction, namely

(**THE** x (& (**DOG1** x) (**HAPPY** x)) (**BARKS1** x))

This will be true only if there is a contextually unique x such that $(\& (\text{DOG1 } x) (\text{HAPPY } x))$ is true, and this x barks.

[Allen 1995: Chapter 8 – Semantics and Logical Form / 237]

Another construct needs to be introduced to handle plural forms, as in a phrase such as *the dogs bark*. A new type of constant called a predicate operator is introduced that takes a predicate as an argument and produces a new predicate. For plurals the predicate operator PLUR will be used. If DOG1 is a predicate that is true of any dog, then $(\text{PLUR } \text{DOG1})$ is a predicate that is true of any set of dogs. Thus the representation of the meaning of the sentence "*The dogs bark*" would be

(THE x : ((PLUR DOG1) x) (BARKS1 x))

Plural noun phrases introduce the possibility of a new form of ambiguity. Note that the natural reading of *The dogs bark* is that there is a specific set of dogs, and each one of them barks. This is called the distributive reading, since the predicate BARKS1 is distributed over each element of the set. In contrast, consider the sentence *The dogs met at the corner*. In this case, it makes no sense to say that each individual dog met; rather the meeting is true of the entire set of dogs. This is called the collective reading. Some sentences allow both interpretations and hence are ambiguous. For instance, the sentence *Two men bought a stereo* can mean that two men each bought a stereo (the distributive reading), or that two men bought a stereo together (the collective reading).

The final constructs to be introduced are the modal operators. These are needed to represent the meaning of verbs such as *believe* and *want*, for representing tense, and for many other constructs. Modal operators look similar to logical operators but have some important differences. Specifically, terms within the scope of a modal operator may have an interpretation that differs from the normal one. This affects what conclusions you can draw from a proposition. For example, assume that Jack is also known as John to some people. There are two word senses that are equal; that is, JACK1 = JOHN22. With a simple proposition, it doesn't matter which of these two constants is used: if (HAPPY JOHN22) is true then (HAPPY JACK1) is true, and vice versa. This is true even in complex propositions formed from the logical operators. If (OR (HAPPY JOHN1) (SAD JOHN1)) is true, then so is (OR (HAPPY JACK1) (SAD JACK1)), and vice versa. The same propositions within the scope of a modal operators such as BELIEVE1, however, are not interchangeable. For instance, if Sue believes that Jack is happy, that is,

(BELIEVE SUE1 (HAPPY JACK1))

then it does not necessarily follow that Sue believes John is happy, that is,

(BELIEVE SUE (HAPPY JOHN22))

because Sue might not know that JACK1 and JOHN22 are the same person. Thus you cannot freely substitute equal terms when they occur within the scope of a modal operator. This is often referred to as the **failure of substitutivity** in modal contexts.

An important class of modal operators for natural language are the tense operators, PAST, PRES, and FUT. So far all examples have ignored the effect of tense. With these new operators, however, you can represent the difference in meaning between *John sees Fido*, *John saw Fido*, and *John will see Fido*, namely as the propositions

(PRES (SEES1 JOHN1 FIDO1))

(PAST (SEES1 JOHN1 FIDO1))

(FUT (SEES1 JOHN1 FIDO1))

You can see that these are modal operators because they exhibit the failure of substitutivity. For example, consider the operator PAST, and assume two constants, say JOHN1 and PRESIDENT1, that are equal now, indicating that John is currently the president. But in the past, John was not the president, so JOHN1 did not equal PRESIDENT!. Given this and the fact that John saw Fido in the past, (PAST (SEES1 JOHN1 FJDO 1)), you cannot conclude that the president saw Fido in the past, that is, (PAST (SEES1 PRESIDENT1 FIDO1)), since John was not the president at that time. Note also that a proposition and its negation can both be true in the past (but at different times). Thus it is possible for both the sentences *John was happy* and *John was not happy* to be true; that is, (PAST (HAPPY JOHN!)) and (PAST (NOT (HAPPY JOHN1))) are both true.

This completes the specification of the basic logical form language. The next sections discuss various extensions that make the language more convenient for expressing ambiguity and capturing semantic regularities.

>> [back](#)

8.4 Encoding Ambiguity in the Logical Form

The previous sections defined many of the constructs needed to specify the logical form of a sentence, and if you were interested solely in the nature of logical form, you could be finished. But representations for computational use have another important constraint on them, namely the handling of ambiguity. A typical sentence will have multiple possible syntactic structures, each of which might have multiple possible logical forms. In addition, the words in the sentence will have multiple senses. Simply enumerating the possible logical forms will not be practical. Rather, we will take an approach where certain common ambiguities can be collapsed and locally represented within the logical form, and we will develop techniques to incrementally resolve these ambiguities as additional constraints from the rest of the sentence and from context are brought into play. Many researchers view this ambiguity encoding as a separate level of representation from the logical form, and it is often referred to as the quasi-logical form.

Perhaps the greatest source of ambiguity in the logical form comes from the fact that most words have multiple senses. Some of these senses have different structural properties, so they can be eliminated given the context of the surrounding sentence. But often words have different senses that have identical structural constraints. At present, the only way to encode these would be to build

[Allen 1995: Chapter 8 – Semantics and Logical Form / 239]

BOX 8.1 The Need for Generalized Quantifiers

For the standard existential and universal quantifiers, there are formulas in standard FOPC equivalent to the generalized quantifier forms. In particular, the formula

(**EXISTS** x : $P_x \ Q_x$)

is equivalent to

$\exists x . P_x \ \& \ Q_x$)

and the universally quantified form

$$(\text{ALL } x : P_x \ Q_x)$$

is equivalent to

$$\check{\exists}x . P_x \wedge Q_x$$

These generalized quantifier forms can be thought of simply as abbreviations. But the other quantifiers do not have an equivalent form in standard FOPC. To see this, consider trying to define the meaning of "*Most dogs bark*" using the standard semantics for quantifiers. Clearly $\check{\exists}x . \text{Dog}(x) \wedge \text{Bark}(x)$ is too strong, and $\forall x . \text{Dog}(x) \wedge \text{Bark}(x)$ is too weak. Could we define a new quantifier M such that some formula of form $Mx . P_x$ has the right truth conditions? Say we try to define $Mx . P_x$ to be true if over half the elements of the domain satisfy P_x . Consider $Mx . \text{Dog}(x) \wedge \text{Bark}(x)$. This will be true only if over half the objects in the domain satisfy $\text{Dog}(s) \wedge \text{Bark}(s)$. But this is unlikely to be true, even if all dogs bark. Specifically, it requires over half the objects in the domain to be dogs! Maybe $Mx . \text{Dog}(x) \vee \text{Bark}(x)$ works. This would be true if more than half the objects in the domain either are not dogs or bark. But this will be true in a model containing 11 objects, 10 seals that bark, and 1 dog that doesn't! You may try other formulas for P_x , but no simple approach will provide a satisfactory account.

a separate logical form for each possible combination of senses for the words in the sentence. To reduce this explosion of logical forms, we can use the same technique used to handle multiple feature values in the syntactic structure. Namely, anywhere an atomic sense is allowed, a set of possible atomic senses can be used. For example, the noun "*ball*" has at least two senses: BALL1, the object used in games, and BALL2, the social event involving dancing. Thus the sentence "*Sue watched the ball*" is ambiguous out of context. A single logical form can represent these two possibilities, however:

1. (THE **b1** : ({BALL1 BALL2} **b1**) (PAST (WATCH1 SUE1 **b1**)))

This abbreviates two possible logical forms, namely

2. (THE **b1** : (BALL1 **b1**) (PAST (WATCH1 SUE1 **b1**)))

and

3. (THE **b1** : (BALL2 **b1**) (PAST (WATCH1 SUE1 **b1**)))

One of the most complex forms of ambiguity in logical forms arises from the relative scoping of the quantifiers and operators. You saw in Section 8.1 that a sentence such as "*Every boy loves a dog*" is ambiguous between two readings depending on the scope of the quantifiers. There is no context-independent method for resolving such issues, so the ambiguity must be represented in the final logical forms for the sentence. Rather than enumerating all possible scopings, which would lead to an exponentially growing number of interpretations based on the number of scoping constructs, we introduce another abbreviation into the logical form language that collapses interpretations together. Specifically, the abbreviated logical form does not contain scoping information at all. Rather, constructs such as generalized quantifiers are treated syntactically like terms and appear in the position indicated by the syntactic structure of the sentence. They are marked using angle brackets to indicate the scoping abbreviation. For example, the logical forms for the sentence "*Every boy loves a dog*" are captured by a single ambiguous form

```
(LOVES1 <EVERY b1 (BOY1 b1) > <A d1 (DOG1 d1) >)
```

This abbreviates an ambiguity between the logical form

```
(EVERY b1 : (BOY1 b1) (A d1 (DOG1 d1) (LOVES1 b1 d1)))
```

and

```
(A d1: (DOG1 d1) (EVERY b1 : (BOY1 b1) (LOVES1 b1 d1)))
```

While the savings don't amount to much in this example, consider that a sentence with four scoping constructs in it would have 24 (4 factorial) possible orderings, and one with five scoping constructs would have 120 orderings. The abbreviation convention allows all these forms to be collapsed to a single representation. Chapter 12 will

explore some heuristic techniques for determining the scope of operators. For the time being, however, it is reasonable to assume that no context-independent scoping constraints need be represented.

If the restriction in a generalized quantifier is a proposition involving a simple unary predicate, a further abbreviation will be used that drops the variable. Thus the form

<EVERY **b1** (BOY **b1**)>

will often be abbreviated as

<EVERY **b1** BOY>

A large number of constructs in natural language are sensitive to scoping. In particular, all the generalized quantifiers, including "*the*", are subject to scoping. For example, in "*At every hotel, the receptionist was friendly*", the preferred reading in almost any context has the definite reference "*the receptionist*" fall within the scope of "*every hotel*"; that is, there is a different receptionist at each hotel.

In addition, operators such as negation and tense are also scope sensitive. For example, the sentence "*Every boy didn't run*" is ambiguous between the reading in which some boys didn't run and some did, that is,

(NOT (EVERY **b1** : (BOY1 **b1**) (RUN1 **b1**)))

and the reading where no boys ran, that is,

(EVERY **b1** : (BOY1 **b1**) (NOT (RUN1 **b1**)))

These two readings are captured by the single logical form

(<NOT RUN1> <EVERY **b1** BOY1>)

where unscoped unary operators (for example, NOT, PAST, PRES, and so on) are wrapped around the predicate.

Finally, let us return to two constructs that need to be examined further: proper names and pronouns. So far we have assumed that each proper name identifies a sense that denotes an object in the domain. While this was a useful abstraction to introduce the basic ideas of logical form, it is not a general enough treatment of the phenomena. In fact, proper names must be interpreted in context, and the name *John* will refer to different people in different situations. Our revised treatment resolves these problems by using a discourse variable that has the property of having the specified name. We will introduce this construct as a special function, namely

(NAME <variable> <name>)

which produces the appropriate object with the name in the current context. Thus, the logical form of "*John ran*" would be (<PAST RUN1> (NAME **j1** "John")).

Arguments similar to those previously given for proper names can be made for pronouns and other indexical words, such as "*here*" and "*yesterday*", and we will treat them using a special function of the form (PRO <variable> <proposition>). For example, the quasi-logical form for "*Every man liked him*" would be

```
(<PAST LIKE1> <EVERY m1 MAN1> (PRO m2 (HE1 m2)) )
```

HE1 is the sense for "*he*" and "*him*", and formally is a predicate true of objects that satisfy the restrictions on any antecedent, that is, being animate-male in this case. As with generalized quantifiers, when the restriction is a simple unary predicate, the pro forms are often abbreviated. For example, the logical form for "*he*" will often be written as (PRO m2 HE1).

The constructs described in this chapter dramatically reduce the number of logical forms that must be initially computed for a given sentence. Not all ambiguities can be- captured by the abbreviations, however, so there will still be sentences that will require a list of possible logical forms, even for a single syntactic structure.

>> [back](#)

8.5 Verbs and States in Logical Form

So far, verbs have mapped to appropriate senses acting as predicates in the logical form. This treatment can handle all the different forms but loses some generalities that could be captured. It also has some annoying properties.

[Allen 1995: Chapter 8 – Semantics and Logical Form / 242]

Consider the following sentences, all using the verb "*break*":

John broke the window with the hammer.

The hammer broke the window.

The window broke.

Intuitively, all these sentences describe the same type of event but in varying detail. Thus it would be nice if the verb "*break*" mapped to the same sense in each case. But there is a problem, as these three uses of the verb seem to indicate verb senses of differing arity. The first seems to be a ternary relation between John, the window, and the hammer, the second a binary relation between the hammer and the window, and the third a unary relation involving the window. It seems you would need three different senses of break, BREAK1, BREAK2, and BREAK3, that differ in their arity and produce logical forms such as

1. (<PAST BREAK1> (NAME **j1** "John") <THE **w1** WINDOW1> <THE **h1** HAMMER1>) ,
2. (<PAST BREAK2> <THE **h1** HAMMER1> <THE **w1** WINDOW1>) , and
3. (<PAST BREAK3> <THE **w1** WINDOW1>)

Furthermore, to guarantee that each predicate is interpreted appropriately, the representation would need axioms so that whenever 1 is true, 2 is true, and whenever 2 is true, 3 is true. These axioms are often called **meaning postulates**. Having to specify such constraints for every verb seems rather inconvenient.

Davidson (1967) proposed an alternative form of representation to handle such cases as well as more complicated forms of adverbial modification. He suggested introducing events into the ontology, and treating the meaning of a sentence like "*John broke it*" along the following lines (translated into our notation):

```
(j e1 : (BREAK e1 (NAME j1 "John") (PRO i1 IT1)))
```

which asserts that **e1** is an event of John breaking the indicated window. Now the meaning of "*John broke it with the hammer*" would be

```
(j e1 : (& (BREAK e1 (NAME j1 "John") (PRO i1 IT1))  
          (INSTR e1 <THE h1 HAMMER>)))
```

The advantage is that additional modifiers, such as *with the hammer* or *on Tuesday* or *in the hallway*, can be incrementally added to the basic representation by adding more predication involving the event. Thus only one sense of the verb *break* need be defined to handle all these cases.

Similar intuitions also motivated the development of case grammar in the early 1970s. While case grammar has been abandoned as originally proposed, many of its concepts have continued to influence current theories. One claim that remains influential is that there is a limited set of abstract semantic relationships that can hold between a verb and its arguments. These are often called thematic roles or case roles, and while different researchers have used different sets of roles, the number required has always remained small. The intuition is that "*John*",

"*the hammer*", and "*the window*" play the same semantic roles in each of these sentences. "*John*" is the actor (the agent role), "*the window*" is the object (the theme role), and "*the hammer*" is the instrument (the instrument role) used in the act of breaking. This suggests a representation of sentence meaning similar to that used in semantic networks, where everything is expressed in terms of unary and binary relations. Specifically, using the three thematic roles just mentioned, the meaning of "*John broke the window*" would be

```
(j e (& (BREAK e) (AGENT e (NAME j1 "John")))  
      (THEME e <THE w1 WINDOW>)) )
```

Because such constructs are so common, we introduce a new notation for them. The abbreviated form for an assertion of the form

```
(j e : (& (Event-p e) (Relation1 e obj1) ... (Relationn e Objn)) )
```

will be

```
(Event-p e [Relation1 obj1] ... [Relationn objn]) )
```

In particular, the quasi-logical form for the sentence "*John broke the window*" using this abbreviation is

```
(<PAST BREAK!> e1 [AGENT (NAME j1 "John") ]  
[THEME <THE w1 WINDOW1>])
```

It turns out that similar arguments can be made for verbs other than event verbs. Consider the sentence "*Mary was unhappy*". If it is represented using a unary predicate as

```
(<PAST UNHAPPY> (NAME j1 "Mary"))
```

then how can we handle modifiers, as in "*Mary was unhappy in the meeting*"? Using the previous argument, we can generalize the notion of events to include states and use the same technique. For instance, we might make UNHAPPY a predicate that asserts that its argument is a state of unhappiness, and use the THEME role to include John, that is,

```
(j s <PAST UNHAPPY> s) (THEME s (NAME j1 "Mary"))
```

Of course, we could use the same abbreviation convention as previously defined for events and thus represent "*Mary was unhappy in the meeting*" as

```
(<PAST UNHAPPY> s [THEME (NAME j1 "Mary") ]  
[IN-LOC <THE m1 MEETING>])
```

It has been argued that having event variables as arguments to atomic predicates is still not enough to capture the full expressiveness of natural language. Hwang and Schubert (1993a), for instance, allow events to be defined by arbitrary sentences. For now, having events and states will be sufficient to develop the important ideas in this book.

[Allen 1995: Chapter 8 – Semantics and Logical Form / 244]

In many situations using explicit event and state variables in formulas is cumbersome and interferes with the development of other ideas. As a result, we will use different representations depending on what is best for the presentation. For example, the logical form of "*Mary sees John*" will sometimes be written as

```
(PRES (SEES1 11 [AGENT (NAME j1 "Mary")]  
[THEME (NAME m1 "John")]))
```

which of course is equivalent to

```
(PRES (j 11 (& (SEES1 11) (AGENT 11 (NAME j1 "Mary")))  
[THEME 11 (NAME m1 "John")))))
```

Other times it will be written in predicate argument form:

```
(PRES (SEES1 (NAME j1 "Mary") (NAME m1 "John")))
```

There has been considerable debate in the literature about whether the thematic role analysis is necessary, or whether all the properties we want from thematic roles can be obtained in other ways from a predicate-argument style representation. While an argument that thematic roles are necessary is hard to construct, the representation is sufficiently helpful to be used in many semantic representation languages.

>> [back](#)

8.6 Thematic Roles

This section examines theories based on the notion of thematic roles, or cases. One motivating example from the last section included the sentences

John broke the window with the hammer.

The hammer broke the window.

The window broke.

"*John*", "*the hammer*", and "*the window*" play the same semantic roles in each of these sentences. "*John*" is the actor, "*the window*" is the object, and "*the hammer*" is the instrument used in the act of breaking of the window. We introduced relations such as AGENT, THEME, and INSTR to capture these intuitions. But can we define these relations more precisely, and what other thematic roles have proved useful in natural language systems? These issues are explored in this section.

Perhaps the easiest thematic role to define is the AGENT role. A noun phrase fills the AGENT role if it describes the instigator of the action described by the sentence. Further, this role may attribute intention, volition, or responsibility for the action to the agent described. One test for AGENT-hood involves adding phrases like

"intentionally" or "in order to" to active voice sentences. If the resulting sentence is well formed, the subject NP can fill the AGENT role. The following sentences are acceptable:

John intentionally broke the window.

John broke the window in order to let in some air.

[Allen 1995: Chapter 8 – Semantics and Logical Form / 245]

But these sentences are not acceptable:

*** The hammer intentionally broke the window.**

*** The window broke in order to let in some air.**

Thus the NP "*John*" fills the AGENT role only in the first two sentences.

Not all animate NPs, even in the subject position, fill the AGENT role. For instance, you cannot normally say

*** John intentionally died.**

* **Mary remembered her birthday in order to get some presents.**

Of course, by adding the phrase *intentionally* to the first sentence, you may construct some plausible reading of the sentence ("John killed himself"), but this is a result of modifying the initial meaning of the sentence "John died".

NPs that describe something undergoing some change or being acted upon will fill a role called THEME. This usually corresponds to the syntactic OBJECT and, for any transitive verb X, is the answer to the question "What was Xed?". For example, given the sentence "*The gray eagle saw the mouse*", the NP "*the mouse*" is the THEME and is the answer to the question "What was seen?". For intransitive verbs, the THEME role is used for the subject NPs that are not AGENTS. Thus in "*The clouds appeared over the horizon*", the NP "*the clouds*" fills the THEME role. More examples follow, with the THEME NP in italics:

The rock broke.

John broke ***the rock***.

I gave John ***the book***.

A range of roles has to do with locations, or abstract locations. First, we must make the distinction mentioned earlier between relations that indicate a location or place and those that indicate motion or paths. The AT-LOC relation indicates where an object is or where an event takes place, as in

Harry walked ***on the road***.

The chair is ***by the door***.

On the road describes where the walking took place, while *by the door* describes where the chair is located.

Other phrases describe changes in location, direction of motion, or paths:

I walked **from here to school** yesterday.

It fell **to the ground**.

The birds flew **from the lake along the river gorge**.

There are at least three different types of phrases here: those that describe where something came from (the FROM-LOC role), such as "*from here*"; those that describe the destination (the TO-LOC role), such as "*to the ground*"; and those that describe the trajectory or path (the PATH-LOC role), such as "*along the gorge*".

[Allen 1995: Chapter 8 – Semantics and Logical Form / 246]

These location roles can be generalized into roles over arbitrary state values, called the AT role, and roles for arbitrary state change (the FROM, TO, and PATH roles). Thus AT-LOC is a specialization of the AT role, and so on. You can see other specializations of these roles when you consider the abstract relation of possession:

I threw the ball **to John**. (the TO-LOC role)

I gave a book **to John**. (the TO-POSS role)

I caught the ball **from John**. (the FROM-LOC role)

I borrowed a book **from John**. (the FROM-POSS role)

The box contains a ball. (the AT LOC role)

John owns a book. (the AT POSS role)

Similarly, you might define AT-TIME, TO-TIME, and FROM-TIME roles, as in

I saw the car **at 3 o'clock**. (the AT-TIME role)

I worked **from one until three**. (the FROM-TIME and TO-TIME role)

The roles apply to general state change as well, as with temperature in

The temperature remains **at zero**. (AT VALUE)

The temperature rose **from zero**. (FROM-VALUE)

Thus the notion of general value and change of value along many dimensions seems to be supported by the similarity of the ways of realizing these roles in sentences.

Another role is motivated by the problem that, given the present taxonomy, you cannot easily classify the role of the NP in a sentence such as

John believed that it was raining.

The THEME role is filled with the clause "*that it was raining*", since this is what is believed. "*John*" cannot be an AGENT because there is no intentionality in believing something. Thus you must introduce a new role, called

EXPERIENCER, which is filled by animate objects that are in a described psychological state, or that undergo some psychological process, such as perception, as in the preceding sentence and as in

John saw the unicorn.

Another role is the BENEFICIARY role, which is filled by the animate person for whom a certain event is performed, as in

I rolled on the floor **for Lucy**.

Find **me** the papers!

I gave the book to Jack **for Susan**.

The last example demonstrates the need to distinguish the TO-POSS role (that is, to Jack) from the BENEFICIARY role.

[Allen 1995: Chapter 8 – Semantics and Logical Form / 247]

The INSTR role describes a tool, material, or force used to perform some event, as in

Harry broke the glass **with the telescope**.

The telescope broke the glass.

I used some flour to make a cake.

I made a cake with some flour.

Depending on the verb, the INSTR role sometimes can be used as the surface subject when the AGENT role is not specified. Natural forces are also included in the INSTR category here, although you could argue for a different analysis. Thus the following are also examples of the INSTR role:

The sun dried the apples.

Jack used the sun to dry the apples.

The AGENT and INSTR roles could be combined into a more general role named CAUSAL-AGENT.

Other roles need to be identified before certain sentences can be analyzed. For example, some sentences describe situations where two people perform an act together:

Henry lifted the piano with Jack.

To handle this, you must introduce a role CO-AGENT to account for the PP with Jack.

A more complicated case occurs in sentences involving exchanges or other complex interactions. For example, consider the sentences

Jack paid \$1 to the man for the book.

Jack bought the book from the man for \$1.

These sentences both describe a situation where Jack gives the man \$1 and receives the book in exchange. In the first sentence, however, the \$1 is the THEME and there is no role to account for the book. In the second sentence the situation is reversed: the book is the THEME and \$1 is unaccounted for. To handle these cases you must add a role CO-THEME for the second object in an exchange.

A more general solution to this problem would be to analyze such sentences as describing two events. The primary event is the one you have been considering so far, but a secondary event may be present. In this analysis you might analyze the first sentence as follows (the primary event being Jack paying the dollar, and the secondary being Jack receiving the book):

Jack: AGENT of both PRIMARY and SECONDARY event

\$1: THEME of PRIMARY event

the man: TO-POSS of PRIMARY, FROM-POSS of SECONDARY

Jack: AGENT of both PRIMARY and SECONDARY event

\$1: THEME of PRIMARY event

the book: THEME of SECONDARY event

[Allen 1995: Chapter 8 – Semantics and Logical Form / 248]

Roles and Subroles	Other Common Names	Definition
CAUSAL-AGENT		the object that caused the event
AGENT		intentional causation
INSTR		force/tool used in causing the event
THEME	PATIENT	the thing affected by the event

Roles and Subroles	Other Common Names	Definition
EXPERIENCER		the person involved in perception or a physical/psychological state
BENEFICIARY		the person for whom an act is done
AT		the state/value on some dimension
AT-LOC	LOCATION	current location
AT-POSS	POSSESSOR	current possessor
AT-VALUE		current value
AT-TIME		current time
TO		final value in a state change
TO-LOC	DESTINATION	final location
TO-POSS	RECIPIENT	final possessor
TO-VALUE		final value
FROM		original value in a state change

Roles and Subroles	Other Common Names	Definition
FROM-LOC	SOURCE	original location
FROM-POSS		original possessor
FROM-VALUE		original value
PATH		path over which something travels
CO-AGENT		secondary agent in an action
CO-THEME		secondary theme in an exchange

Figure 8.5 Some possible semantic roles

This possibility will not be pursued further, however, since it leads into many issues not relevant to the remainder of this chapter.

Figure 8.5 provides a summary of most of the roles distinguished thus far and the hierarchical relationships between them.

As you've seen, verbs can be classified by the thematic roles that they require. To classify them precisely, however, you must make a distinction between roles that are "intimately" related to the verb and those that are not. For example, almost any past tense verb allows an AT-TIME role realized by the adverb "*yesterday*". Thus this role is apparently more a property of verb phrases in general than a property of any individual verb. However, other roles – namely, those realized by constituents for which the verb subcategorizes – seem to be properties of the verb. For example, the verb "*put*" subcategorizes for a PP, and furthermore, this PP must realize the TO-LOC role. In verb classification this latter type of role is important, and these roles are called the **inner roles** of the verb.

The preceding examples suggest one test for determining whether a given role is an inner role for a given verb: if the role is obligatory, it is an inner role. Other inner roles, however, appear to be optional, so other tests are also needed. Another test is based on the observation that all verbs may take at most one NP in

any given inner role. If multiple NPs are needed, they must be related by a conjunction. Thus you can say

John and I ran to the store.

but not

*** John I ran to the store.**

Similarly, you can say

I ran to the store and to the bank.

but not

*** I ran to the store to the bank.**

Thus the AGENT and TO-LOC roles for the verb run are inner roles.

Verbs typically specify up to three inner roles, at least one of which must always be realized in any sentence using the verb. Sometimes a particular role must always be present (for example, TO-LOC with put). Typically, the THEME role is also obligatory, whereas the AGENT role is always optional for any verb that allows the passive form.

There are also syntactic restrictions on how various roles can be realized. Figure 8.6 shows a sample of ways that roles can be realized in different sentences.

The following are some sample sentences with each verb in italics and its argument, whether NP, PP, or embedded
5, classified by its role in order of occurrence:

- | | |
|--|--------------------------------|
| Jack ran . | AGENT only |
| Jack ran with a crutch. | AGENT +INSTR |
| Jack ran with a crutch for Susan. | AGENT +INSTR +BENEFICIARY |
| Jack destroyed the car. | AGENT +THEME |
| Jack put the car through the wall. | AGENT +THEME +PATH |
| Jack sold Henry the car. | AGENT +TO-POSS +THEME |
| Henry pushed the car from Jack's house to the junkyard. | AGENT +THEME +FROM-LOC +TO-LOC |
| Jack is tall . | THEME |
| Henry believes that Jack is tall. | EXPERIENCER +THEME |
| Susan owns a car. | AT-POSS +THEME |

Jack ran .	AGENT only
I am in the closet.	THEME +AT-LOC
The ice melted .	THEME
Jack enjoyed the play.	EXPERIENCER + THEME
The ball rolled down the hill to the water.	THEME +PATH +TO-LOC

[Allen 1995: Chapter 8 – Semantics and Logical Form / 250]

Role	Realization
AGENT	as subject in active sentences

Role	Realization
	preposition <i>by</i> in passive sentences
THEME	as object of transitive verbs
	s subject of nonaction verbs
INSTR	as subject in active sentences with no agent preposition <i>with</i>
EXPERIENCER	as animate subject in active sentences with no agent
BENEFICIARY	as indirect object with transitive verbs preposition <i>for</i>
AT-LOC	prepositions <i>in</i> , <i>on</i> , <i>beyond</i> , etc.
AT-POSS	possessive NP
	as subject of sentence if no agent
TO-LOC	prepositions <i>to</i> , <i>into</i>

Role	Realization
TO-POSS	preposition <i>to</i> , indirect object with certain verbs
FROM-LOC	prepositions <i>from</i> , <i>out of</i> , etc.
FROM-POSS	preposition <i>from</i>

Figure 8.6 Common realizations of the major roles

>> [back](#)

8.7 Speech Acts and Embedded Sentences

Sentences are used for many different purposes. Each sentential mood indicates a different relation between the speaker and the propositional content of the utterance. These issues will be examined in greater detail in later chapters. For now the logical form language is extended to capture the distinctions. Each of the major sentence types has a corresponding operator that takes the sentence interpretation as an argument and produces what is called a **surface speech act**. These indicate how the proposition described is intended to be used to update the discourse situation. They are indicated by new operators as follows:

ASSERT - the proposition is being asserted.

Y/N-QUERY - the proposition is being queried.

COMMAND - the proposition describes an action to perform.

WH-QUERY - the proposition describes an object to be identified.

For declarative sentences, such as "*The man ate a peach*", the complete LF is

```
(ASSERT (<PAST EAT> e1 [AGENT <THE m1 MAN1]  
[THEME <A p1 PEACH1>] ))
```

For yes/no questions, such as "*Did the man eat a peach?*", the LF is

[Allen 1995: Chapter 8 – Semantics and Logical Form / 251]

```
(Y/N-QUERY (<PAST EAT> e1 [AGENT <THE m1 MAN1>  
[THEME <A p1 PEACH1>] ))
```

For commands, such as "*Eat the peach*", the LF is

```
(COMMAND (EAT e1 [THEME <THE p1 PEACH1>] ))
```

For wh-questions, such as "*What did the man eat?*", several additions need to be made to the logical form language. First, you need a way to represent the meaning of noun phrases involving wh-terms. A new quantifier WH is defined that indicates that the term stands for an object or objects under question. Thus a noun phrase such as "what" would be represented <WH **o1** ANYTHING>, "which man" as <WH **m1** MAN1>, and "who" as <WH **p1** PERSON>. Finally, for question forms such as "how many" and "how much", we introduce the quantifiers HOW-MANY and HOW-MUCH. Note that wh-terms are scope sensitive and thus treating them as quantifiers makes sense. The question "*Who is the leader of every group?*" is ambiguous between asking for a single person who leads every group, and asking for the leader of each of the groups.

Thus, the logical form of the sentence "*What did the man eat?*" is

```
(WH-QUERY (<PAST EAT> e1 (AGENT <THE m1 MAN1>  
[THEME <WH w1 PHYSOBJ>]))
```

Embedded sentences, such as relative clauses, end up as complex restrictions within the noun phrase construction and thus do not need any new notation. For example, the logical form of the sentence "*The man who ate a peach left*" would be

```
(ASSERT  
<PAST LEAVE> 11  
[AGENT <THE m1 (& (MAN1 m1)  
<PAST EAT1> e2  
[AGENT m1]  
[THEME <A p1 PEACH>])>])
```

Figure 8.7 gives a formal definition of the syntax of the logical form language introduced throughout this chapter.

Figure 8.8 gives the additional rules defining the syntax for the quasi-logical form language.

>> [back](#)

8.8 Defining Semantic Structure: Model Theory

So far, the term semantics has been used to reflect the representation of meanings for sentences. This was expressed in the logical form language. There is another meaning of semantics used in formal language theory that provides a meaning of the logical form language itself. It concentrates on distinguishing the different classes of semantic objects by exploring their model-theoretic properties, that is, by defining semantic units in terms of their mapping to set theory. While these techniques are most commonly used for logics, they can be applied to natural language as well. This section is optional and is not necessary for understanding

[Allen 1995: Chapter 8 – Semantics and Logical Form / 252]

$UTTERANCE \rightarrow (\text{ASSERT PROPOSITION}) \mid$
 $(\text{Y/N-QUERY PROPOSITION}) \mid$
 $(\text{COMMAND PROPOSITION}) \mid$
 $(\text{WH-QUERY PROPOSITION})$

$PROPOSITION \rightarrow (n\text{-ARY-OPERATOR } PROPOSITION_1 \dots PROPOSITION_n) \mid$
 $(\text{QUANTIFIER VARIABLE} : PROPOSITION \text{ PROPOSITION}) \mid$
 $(n\text{-ARY-PREDICATE TERM}_1 \dots TERM_n) \mid$
 $(\text{EVENT-STATE-PRED VARIABLE} [\text{ROLE-NAME TERM}]_1 \dots$
 $[\text{ROLE-NAME TERM}]_n)$

$TERM \rightarrow \text{VARIABLE} \mid$
 $(\text{NAME VARIABLE NAME-STRING}) \mid$
 $(\text{PRO VARIABLE PROPOSITION})$

$1\text{-ARY-OPERATOR} \rightarrow \text{NOT} \mid \text{PAST} \mid \text{PERF} \mid \text{PROG} \mid \dots$

$2\text{-ARY-OPERATOR} \rightarrow \text{AND} \mid \text{BUT} \mid \text{IF-THEN} \mid \dots$

$QUANTIFIER \rightarrow \text{THE} \mid \text{SOME} \mid \text{WH} \mid \exists \mid \dots$

$VARIABLE \rightarrow \text{b1} \mid \text{man3} \mid \dots$

$1\text{-ARY-PREDICATE} \rightarrow \text{TYPE-PREDICATE} \mid \text{HAPPY1} \mid \text{RED1} \mid \dots$

$\text{TYPE-PREDICATE} \rightarrow \text{EVENT-STATE-PRED} \mid (\text{PLUR TYPE-PREDICATE}) \mid \text{MAN1} \mid \dots$

$\text{EVENT-STATE-PRED} \rightarrow \text{RUN1} \mid \text{LOVE3} \mid \text{GIVE1} \mid \text{HAPPY} \mid \dots$

$2\text{-ARY-PREDICATE} \rightarrow \text{ROLE-NAME} \mid \text{ABOVE1} \mid \dots$

$ROLE-NAME \rightarrow \text{AGENT} \mid \text{THEME} \mid \text{AT-LOC} \mid \text{INSTR} \mid \dots$

$NAME-STRING \rightarrow \text{"John"} \mid \text{"The New York Times"} \mid \dots$

Figure 8.7 A formal definition of the syntax of the logical form language

Figure 8.7 A formal definition of the syntax of the logical form language

TERM → <*QUANTIFIER VARIABLE PROPOSITION*>
TERM → <*n-ARY-OPERATOR TERM₁ ... TERM_n*>
n-ARY-PREDICATE
 → <*m-ARY-OPERATOR n-ARY-PREDICATE₁ ... n-ARY-PREDICATE_m*>
n-ARY-OPERATOR → { *n-ARY-OPERATOR₁ ... n-ARY-OPERATOR_m* }
QUANTIFIER → { *QUANTIFIER₁ ... QUANTIFIER_m* }
n-ARY-PREDICATE → { *n-ARY-PREDICATE₁ ... n-ARY-PREDICATE_m* }
TYPE -PREDICATE → { *TYPE -PREDICATE₁ ... TYPE -PREDICATE_m* }
EVENT-STATE-PRED → { *EVENT-STATE-PRED₁ ... EVENT-STATE-PRED_m* }
ROLE-NAME → { *ROLE-NAME₁ ... ROLE-NAME_m* }

Figure 8.8 Additional rules defining the quasi-logical form

Figure 8.8 Additional rules defining the quasi-logical form

the rest of the book. If you are not familiar with FOPC and its model-theoretic semantics, you should read Appendix B before reading this section.

[Allen 1995: Chapter 8 – Semantics and Logical Form / 253]

The basic building block for defining semantic properties is the idea of a **model**, which informally can be thought of as a set of objects and their properties and relationships, together with a specification of how the language being studied relates to those objects and relationships. A model can be thought of as representing a particular context in which a sentence is to be evaluated. You might impose different constraints on what properties a model must have. For instance, the standard models for logic, called Tarskian models, are complete in that they must map every legal term in the language into the domain and assign every statement to be true or false. But various forms of partial models are possible that do not assign all statements to be either true or false. Such models are like the situations described in the last section, and in fact a mathematical theory of situations could be a formal model in the general sense we are using here. These issues will be discussed further as they become relevant to the discussion.

Model theory is an excellent method for studying context-independent meaning, because the meanings of sentences are not defined with respect to one specific model but rather by how they relate to any possible model. In other words, the meaning of a sentence is defined in terms of the properties it has with respect to an arbitrary model.

Formally, a model m is a tuple $\langle D_m, I_m \rangle$, where D_m is the domain of interpretation (that is, a set of primitive objects), and I is the interpretation function. To handle natural language, the domain of interpretation would have to allow objects of all the different types of things that can be referred to, including physical objects, times, Locations, events, and situations. The interpretation function maps senses and larger structures into structures defined on the domain. For example, the following describe how an interpretation function will interpret the senses based on some lexical classes:

Senses of noun phrases - refer to specific objects; the interpretation function maps each to an element of D_m .

Senses of singular common nouns (such as "*dog*", "*idea*", "*party*") - identify classes of objects in the domain; the interpretation function maps them to sets of elements from D_m (that is, subsets of D_m).

Senses of verbs - identify sets of n-ary relations between objects in D . The arity depends on the verb. For instance, the exercising sense of "run", RUN1, might map to a set of unary relations ($\langle x \rangle$, where x runs), and the usual sense of loves, LOVES 1, to a set of binary relations ($\langle x, y \rangle$, where x loves y), and the usual sense of "put", PUT1, to a set of ternary relations ($\langle x, y, l \rangle$, where x puts y in location l).

We can now define a notion of **truth** of a proposition in the logical form language, again relative to an arbitrary model m . A proposition of the form $(v_n \ a_1 \ \dots \ a_n)$ is true with respect to a model in if and only if the tuple

consisting of the interpretations of the a_i 's is in the set that is the interpretation of v_n . i.e., the

[Allen 1995: Chapter 8 – Semantics and Logical Form / 254]

tuple $\langle I_m(a_1), \dots, I_m(a_n) \rangle$ is in the set $I_m(v_n)$. Following conventional notation, we will write the fact that a proposition P is true with respect to a model m by

$m \models P$

This is sometimes also read as "m supports P". For example, m supports (RUN1 JACK1) only if $I_m(JACK1)$ is in the set $I_m(RUN1)$, and m supports (LOVES1 JACK1 SUE1) only if $\langle I_m(JACK1), I_m(SUE1) \rangle$ is in the set $I_m(LOVES1)$.

The semantics of negation depend on the properties of the models used. In standard Tarskian semantics, where the models are complete, negation is defined in terms of the absence of the relation necessary to make the proposition true; that is, a model m supports (NOT P) if and only if it doesn't support P. In other models this may be too strong, as a proposition might be neither true nor false. Such models would have to maintain two disjoint sets for every relation name: the tuples for which the relation holds and the tuples for which it is false. Any tuple not in either of these sets would be neither true nor false.

The semantics of the quantifiers in FOPC is fairly simple. For example, the proposition $\exists \mathbf{x} . \ P(\mathbf{x})$ is true with respect to a model \mathbf{m} if and only if $P(\mathbf{x})$ is true for any value of \mathbf{x} in D_m . The proposition $\forall \mathbf{x} . \ P(\mathbf{x})$, on the other hand, is true with respect to a model \mathbf{m} if and only if $P(\mathbf{x})$ is true for at least one value of \mathbf{x} in D_m . For natural languages we have generalized quantifiers. The truth conditions for each quantifier specify the required relationship between the objects satisfying the two propositions. For example, consider the proposition $(\text{MOST1 } \mathbf{x} \ (P \ \mathbf{x}) \ (Q \ \mathbf{x}))$. This might be defined to be true with respect to a model \mathbf{m} if and only if over half the objects in $I_m(P)$ are in $I_m(Q)$. For example, $(\text{MOST1 } \mathbf{x} \ (\text{DOG1 } \mathbf{x}) \ (\text{BARKS1 } \mathbf{x}))$ would be true with respect to a model \mathbf{m} if and only if more than half the elements of the set $\{\mathbf{x} \mid (\text{DOG1 } \mathbf{x})\}$ are also in the set $\{\mathbf{y} \mid (\text{BARKS1 } \mathbf{y})\}$, that is, only if over half the dogs in \mathbf{m} bark.

Semantic Relations Among Sentences

With a semantic theory in hand, it is now possible to be more precise about certain inferential relationships among sentences. For instance, when you know that some sentence S is true, then some other sentences must also be true. For instance, if you know "*A red box is on the table*" then you also know that "*A box is on the table*". This relationship between sentences is called entailment, and we say the sentence "*A red box is on the table*" entails the sentence "*A box is on the table*". Formally, entailment can be defined in terms of the models that support the

sentences. In particular, sentence S entails sentence S' if and only if every model that supports S also supports S' ; that is,

S entails S' if and only if for any model m , if $m \models S$ then $m \models S'$

Conversely, if there are no models that support both sentences simultaneously, the sentences are said to contradict each other; that is, there is no

[Allen 1995: Chapter 8 – Semantics and Logical Form / 255]

possible situation in which both statements can be true. Slightly modifying the previous example, we know that the sentences "*A red box is on the table*" and "*There is no box on the table*" are contradictory because there is no model that can support these two sentences simultaneously.

Given that entailments must hold in all possible models, it might seem that there won't be very many entailments to be found. This turns out not to be the case as the models share more properties than you might first expect. In particular, all the context-independent properties of language will be shared by every model. Consider the above example. While the set of boxes and the set of red things could vary dramatically from model to model, the above entailment did not rely on the actual objects being referred to. Rather, it depended only on the general interpretation of the senses of "*red*" and "*box*", and the set-theoretic property that given two sets X and Y , then the

intersection of X and Y is contained in the set X (and Y for that matter). Thus, as long as the model interprets the NP "*A red box*" as selecting an object from the set RED1 intersected with the set BOX1, then that same object will be in the set BOX1. This is all that is required. Thus every model will include the entailment. Furthermore, Section 8.2 suggested that word senses could be organized into a hierarchy based on set inclusion. This defines an additional context-independent structure on the senses and thus will be included in all possible models. As a consequence, the sentence "*A mare ran away*" will entail the sentence "*A horse ran away*", because every model must map MARE1 to a subset of HORSE1.

While entailment is a useful property, natural language understanding typically deals in weaker connections between sentences, which we will call implications. A sentence S implies a sentence S' if S being true suggests or makes it likely that S' is also true. For example, the sentence "*I used to walk to school every day*" implies that I don't walk to school anymore but doesn't entail it, as I could say "*I used to walk to school every day. In fact, I still do!*" Given that these two sentences are not anomalous, there must be a model that assigns truth to both of them, and in this model it will not be true that I don't walk to school anymore. However, in a typical context, the models that are relevant to that context might all support that I don't walk to school anymore. One of the difficult problems in language understanding results from the fact that most inferences made are implications rather than entailments, and thus conclusions drawn at one time might need to be reconsidered and retracted at a later time.

>> [back](#)

Modal Operators and Possible Worlds Semantics

To provide a semantics for modal operators, more structure must be imposed on the model theory. In particular, the truth of a property will be defined with respect to a set of models rather than a single model. In addition, this set of models may have a rich structure and hence is called a model structure.

[Allen 1995: Chapter 8 – Semantics and Logical Form / 256]

BOX 8.2 Extensional and Intensional Readings

The semantic framework discussed in this chapter almost exclusively concerns what is called the extensional meaning of expressions. This is a way of saying that meaning is defined in terms of what the expressions denote in the world. For example, the word sense **DOG1** obtains its meaning from the set of objects that it denotes, namely **I_m(DOG1)**. Likewise, a term such as "*the dog*" or "*John*" would derive its meaning from the object in the domain that it denotes. If two terms denote the same object, then they are semantically indistinguishable. The extensional view covers a wide range of language but cannot account for many expressions. Consider a classic example from Montague (1974):

The temperature is rising.

What is the meaning of the NP "*the temperature*" in this sentence? If it denotes a particular value, say 30°, then it could not be rising, as the sentence "30° *is rising*" is nonsensical. Rather, the phrase "*the temperature*" must denote a function over time, say the function **Temp** that, given a time and a location, yields the temperature. Given this meaning, the predicate "*is rising*" would be true of any function that is increasing in value over time. Thus there seem to be two different meanings for the term "*the temperature*". The extensional reading would be the actual value at a particular time and place, say 30°, while the intensional reading would be a function that takes a time and place and yields a value. Some verb phrases, such as "*is rising*", require the intensional meaning, while others, such as in the sentence "*The temperature is over 90°*", require the extensional meaning, as it would make no sense to say that a function has the property of being over 90°.

Because of the complexities in dealing with intensional readings, we will only deal with extensional readings of expressions throughout this book.

Consider the past tense operator PAST. Let us assume that each individual model \mathbf{m} represents the state of the world at some point in time. Then a model structure would be the set of models describing one possible history of the world, in which models are related to each other by an operator $<$, where $\mathbf{m}_1 < \mathbf{m}_2$ means \mathbf{m}_1 describes a state of the world before the state described by \mathbf{m}_2 . The truth of propositions is now defined with respect to an arbitrary model structure \mathbf{D} that has the $<$ relation between models. For example, we can now define a semantics for the expression **(PAST P)** with respect to a model \mathbf{m} in a model structure \mathbf{D}

$\mathbf{m} \models_{\mathbf{D}} (\text{PAST } \mathbf{P}) \text{ if and only if there is another model } \mathbf{m}' \text{ in } \mathbf{D} \text{ such that}$
 $\mathbf{m}' < \mathbf{m} \text{ and } \mathbf{m}' \models_{\mathbf{D}} \mathbf{P}$

In other words, **(PAST P)** is true in a model in if and only if there is another model in the model structure that describes a time previous to \mathbf{m} in which \mathbf{P} is true. This type of analysis is called a possible worlds semantics.

>> [back](#)

[Allen 1995: Chapter 8 – Semantics and Logical Form / 257]

Summary

This chapter presented a context-independent semantic representation called the logical form. Such a representation is desirable to simplify the process of computing semantic structures from the syntactic structure, and more generally to modularize the processing of sentences by separating out contextual effects. The logical form language uses many of the concepts from FOPC, including terms, predicates, propositions, and logical operators. The most significant extensions to basic FOPC were the following:

- generalized quantifiers, which indicate a relationship between two sets and correspond to words such as *each*, *every*, *some*, *most*, *several*, and so on
- modal operators, which identify different modalities in which to consider propositions and correspond to words such as *believe* and *hopes*, as well as the tense operators
- predicate operators, which map one predicate into a new predicate, as with the operator for plural forms that takes a predicate describing individuals with a property P and produces a predicate that describes sets of individuals, each with property P

In addition, the logical form language can encode many common forms of ambiguity in an efficient manner by allowing alternative senses to be listed wherever a single sense is allowed.

An important aspect of the logical form language is its use of event and state variables on predicates. This allows you to represent additional adverbial modifiers without having to introduce different predicates for each possible combination of arguments to the verb. A representation based on thematic roles was also introduced. While the roles may differ from system to system, this style of representation is very common in natural language systems.

>> [back](#)

Related Work and Further Readings

The logical form language developed here draws from many sources. The most influential early source is the representation language in the LUNAR system (Woods, 1978), which contains, initial ideas for many of the details in this representation. More recent strong influences are the logical form languages developed by Schubert and Pelletier (1982), Hwang and Schubert (1993a), and Moore (1981), and the quasi-logical form in the Core Language Engine (Alshawi, 1992).

Like many logical form languages, the one developed here is heavily based on techniques of introducing events into the ontology to handle optional arguments and other verb modifiers. This technique became influential following a landmark paper by Davidson (1967) who presented many arguments for the approach. Moore (1981), Alshawi (1992), and Hobbs et al. (1987) are all good examples of such event-based systems.

Work on thematic roles originates from work on case grammar by Fillmore (1968), who introduced six cases. These ideas have been adapted by many researchers since in linguistics and philosophy (for example, see Jackendoff (1972), Dowty (1989), and Parsons (1990)). Similar techniques can be found in many computational systems. In these systems, thematic roles are often linked to the underlying knowledge representation (for example, Charniak (1981)).

The idea of encoding scoping ambiguity in the logical form has been used by many systems. Woods (1978) and Cooper (1983), for instance, suggested a two-part representation of logical form: one part for the predicate-argument structure and the other for the quantifier information. Encoding the scoped forms using a special syntax has been used in Schubert and Pelletier (1982), McCord (1986), Hobbs and Shieber (1987), and many recent systems such as the Core Language Engine (Alshawi, 1992). This encoding is not only important for encoding ambiguity, but also plays a key role in allowing the definition of compositional semantic interpretation theories, as will be discussed in Chapter 9.

An excellent general source on semantics in linguistics can be found in Chierchia and McConnell-Ginet (1990). This book contains discussions on the nature of semantics and issues such as ambiguity and vagueness, indexicality, denotational semantics, generalized quantifiers, intensionality, and a host of other issues. McCawley (1993) is also an excellent and accessible source on the logical underpinnings of semantics. More detailed presentation of the mathematical underpinnings of semantic representations can be found in Partee et al. (1993). Generalized quantifiers are discussed in Barwise and Cooper (1981). Many of the semantic ideas discussed here were first developed within symbolic logic, for which there are many excellent texts (for example, Thomason

(1970); Barwise and Etchemendy (1987)). The current form of semantics for natural language is most heavily influenced by the work of Montague (1974).

Much of the current work in formal semantics is within the framework of situation semantics. The notion of a situation discussed in this chapter is drawn from that work. Situation semantics was developed by Jon Barwise and collaborators, and an initial version of the theory is described in Barwise and Perry (1983). An excellent discussion and formalization of situations is found in Devlin (1991).

>> [back](#)

Exercises for Chapter 8

(easy) State why each of the following sentences are ambiguous or not. Specifically, state whether they are ambiguous because of their possible syntactic structures, their word senses, their semantic structures, or a combination of these factors. Give a paraphrase of each reading.

A man stopped at every truck stop.

Several people ate the pizza.

We saw her duck.

[Allen 1995: Chapter 8 – Semantics and Logical Form / 259]

2. (medium) For each of the following words, argue whether the word is ambiguous or vague. Give at least one linguistic test that supports your answer. Discuss any difficulties that arise in making your decision.

face - as a noun, as in what's on your head, the front of a clock, and the side of a mountain

bird - as a noun, as in animals that fly (such as a sparrow) and china figures of such animals

record - as a noun, as in where a school keeps your grades or where the FBI writes down your movements

3. (easy) Identify the senses of the word **can** used in the following sentences, and specify whether the sense is a term, property (unary predicate), n-ary predicate, logical operator, generalized quantifier, predicate operator, or modal operator. Justify your classification and discuss alternate interpretations that could work just as well. Give an example logical form that uses each sense.

The yellow can fell to the ground.

He can see it.

He wants to can the tomatoes.

4. (easy) Expand out the following ambiguous logical forms, giving a complete list of the full logical forms allowed by the ambiguous form.

```
({RUN1 RUN2} [AGENT (PRO h1 HE1)])  
  
(SEES1 s1 [AGENT (PRO h1 HE1)]  
  
[<EVERY h1 {BALL1 BALL2}>])  
  
(GIVES1 l1 [AGENT <EVERY m1 MAN1>]  
  
[THEME <A g1 GIFT1>])  
  
  
[AGENT <EVERY m1 (& (MAN1 m1) (HAPPY m1))>])
```

5. (medium) Specify a quasi-logical form for the following sentences. If the sentence is ambiguous, make sure you represent all the possibilities, either using ambiguous logical forms or by listing several logical forms.

George ate a pizza at every road stop.

Several employees from every company bought a pizza.

We saw John in the park by the beach.

6. (medium) Specify the roles for each NP in the following sentences. Give a plausible logical form for each sentence.

We returned the ring to the store.

We returned to the party.

The owner received a ticket.

[Allen 1995: Chapter 8 – Semantics and Logical Form / 260]

7. (medium)

a. Specify a formal semantic model that makes each of the following logical forms true.

(IN1 JOHN1 ROOM1)

(BOY1 JOHN1)

(EVERY **b1** : BOY1 (EAT **b1** PLZZA1))

Specifically, define a set of objects for the domain of the model, and specify the interpretation function that maps each of these symbols into an element or set in the domain. Demonstrate that each of these logical forms is assigned true by specifying their interpretation.

b. Does your model also assign true to

(EAT JOHN1 PIZZA1)

Could you build a model that does the opposite (assigns false if yours assigns true, or vice versa)? If so, give the model. If not, why not?

8. (medium) For each of the following lists of sentences, state whether the first sentence entails or implies each of the sentences that follow it or that there is no semantic relationship among them. Justify your answers using some linguistics tests.

a. John didn't manage to find the key.

John didn't find the key.

John looked for the key.

The key is hard to find.

b. John was disappointed that Fido was last in the dog show.

Fido was last in the dog show.

Fido was entered in the dog show.

John wanted Fido to win.

Fido is a stupid dog.

>> [back](#)

Allen 1995 : Natural Language Understanding

	<u>Contents</u>	<u>Preface</u>	<u>Introduction</u>	
<u>previous chapter</u>	<u>Part I Syntactic Processing</u>	<u>Part II Semantic Interpretation</u>	<u>Part III - Context / World Knowledge</u>	next chapter
	<u>Appendices</u>	<u>Bibliography</u>		
	<u>Summaries</u>	<u>Further Readings</u>	<u>Exercises</u>	

Chapter 9 : Linking Syntax and Semantics

[9.1 Semantic Interpretation and Compositionality](#)

[9.2 A Simple Grammar and Lexicon with Semantic Interpretation](#)

[9.3 Prepositional Phrases and Verb Phrases](#)

[9.4 Lexicalized Semantic Interpretation and Semantic Roles](#)

[9.5 Handling Simple Questions](#)

[9.6 Semantic Interpretation Using Feature Unification](#)

[9.7 Generating Sentences from Logical Form](#)

[Summary](#)

[Related Work and Further Readings](#)

9.1 Semantic Interpretation and Compositionality

Exercises for Chapter 9

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 263]

This chapter discusses a particular method for linking logical forms with syntactic structures. This will allow logical forms to be computed while parsing, a process called semantic interpretation. One version discussed also allows a syntactic tree to be generated from a specified logical form, a process called semantic realization. To fully couple syntax and semantics, there must be a well-formed meaning expression for every constituent. The relation between the meaning of a constituent and the meanings of its subconstituents can then be specified in the grammar using features. Because each syntactic rule has a corresponding semantic interpretation rule, this method is often referred to as a rule-by-rule style of semantic interpretation.

Section 9.1 discusses the notion of compositionality and introduces the lambda calculus as a tool for building compositional theories. Sections 9.2 and 9.3 then examine some basic constructs in language and develop a grammar for a small fragment of English that computes the logical form of each constituent as it is parsed. The logical form used in Sections 9.2 and 9.3 is a predicate argument structure. Section 9.4 shows how to generate a logical form using semantic roles and briefly discusses the need for hierarchical lexicons to reduce the amount of work required to specify the meanings of lexical items. Section 9.5 discusses how semantic interpretation relates to gaps and shows how to handle simple questions. Section 9.6 develops an alternative method of computing logical forms that uses additional features rather than lambda expressions, thus allowing us to express reversible grammars. Optional Section 9.7 discusses semantic realization, showing how to generate a sentence given a logical form and a reversible grammar.

>> [back](#)

9.1 Semantic Interpretation and Compositionality

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 271]

In linguistics compositionality is often defined in terms of a strict criterion:

One subconstituent must have as its meaning a function that maps the meanings of the other subconstituents into the meaning of the new constituent. In computational approaches, the requirement is often more relaxed and requires a mere mental process of building meanings constituent by constituent, where the

[Allen 1995: Chapter 9 – Linking Syntax and Semantics / 264]

meaning of a constituent is produced by some well-defined computational function (that is, a program), that uses the meanings of the subconstituents as inputs.

Compositional models tend to make grammars easier to extend and maintain. But developing a compositional theory of semantic interpretation is harder than it looks. First, there seems to be a structural inconsistency between syntactic structure and the structure of the logical forms. For instance, a classic problem arises with quantified sentences. Consider the sentence *Jill loves every dog*. The syntactic structure of this sentence clusters the words into phrases in the obvious way: ((Jill) (loves (every dog))). But the unambiguous logical form of the sentence in a predicate-argument form would be something like

```
(EVERY d : (DOG1 d) (LOVES1 11 (NAME j1 "Jill") d))
```

This asserts that for every dog *d* there is an event *l1* that consists of Jill loving *d*. There seems to be no simple one-to-one correspondence between parts of the logical form and the constituents in syntactic analysis. The phrase *every dog*, for instance, is a subconstituent of the VP *loves every dog*. Its semantic interpretation, however - the generalized quantified phrase (EVERY *d* : (DOG1 *d* ...)) - seems to have the meaning of the verb phrase as a part of it. Worse than that, the interpretation of *every dog* seems to be split - it produces both the quantifier structure outside the predicate as well as an argument to the predicate. As a result, it is hard to see how the meaning of *every dog* could be represented in isolation and then used to construct the meaning of the sentence. This is one of the most difficult problems to be dealt with if we are to have a compositional theory. Note that introducing the unscoped logical form constructs provides one way around the problem. If we define the goal of semantic interpretation as producing an unscoped logical form, the unscoped version of this sentence would be

```
(LOVES1 11 (NAME j1 "Jill") <EVERY d DOG1>)
```

which is much closer in structure to the syntactic form.

Another challenge to compositional theories is the presence of idioms. For instance, you may say *Jack kicked the bucket*, meaning that Jack died. This interpretation seems to have no relation to the meanings of the verb *kick* and

nothing to do with buckets. Thus the meaning of the sentence does not seem to be constructed out of the meaning of its subconstituents. One way to handle such cases is to allow semantic meanings to be assigned to entire phrases rather than build the meaning compositionally. So far, we have assumed that the primitive unit is the word (or morpheme). Idiomatic expressions suggest that this might be generalized so that complete phrases may have a primitive (that is, nonderived) meaning. In the previous example the verb phrase *kick the bucket* has a primitive meaning similar to the verb *die*. This approach is supported by the observation that certain syntactic paraphrases that are fine for sentences given their compositional meaning do not apply for idiomatic readings. For instance, the passive sentence *The bucket was kicked by Jack* could not mean that Jack died.

[Allen 1995: Chapter 9 – Linking Syntax and Semantics / 265]

Interestingly, *Jack kicked the bucket* is ambiguous. It would have one meaning constructed compositionally from the meanings of each of the words, say (KICK1 k1 (NAME j1 "Jack") <THE b BUCKET>), and another constructed from the meaning of the word *Jack* and the primitive meaning of the phrase *kick the bucket*, say (DIE1 d1 (NAME j1 "Jack")).

Another approach to this problem is to introduce new senses of words that appear in idioms. For example, there might be a sense of *kick* that means DIE1, and that subcategorizes for an object of type BUCKET1. While idioms are a very interesting and important aspect of language, there will not be the space to deal with them in the next

few chapters. For the purposes of this book, you can assume that the primitive meanings are always associated with the word.

If the process of semantic interpretation is compositional, then you must be able to assign a semantic structure to any syntactic constituent. For example, you must be able to assign some uniform form of meaning to every verb phrase that can be used in any rule involving a VP as a subconstituent. Consider the simplest case, where the VP consists of an intransitive verb, as in a sentence such as *Jack laughed*. One suggestion is that the meaning of the verb phrase *laughed* is a unary predicate that is true of any object that laughed in the past. Does this approach generalize? In other words, could every VP have a meaning that is a unary predicate? Consider the sentence *Jack kissed Sue*, with the logical form

```
(KISS1 k1 (NAME j1 "Jack") (NAME s1 "Sue")) )
```

What is the meaning of the VP *kissed Sue*? Again, it could be a unary predicate that is true of any object that kissed Sue. But so far we have no way to express such complex unary predicates. The lambda calculus provides a formalism for this. In particular, the expression

```
( $\hat{U}$  x (1551 k1 x (NAME s1 "Sue"))))
```

is a predicate that takes one argument. You can view x as a parameter, and this predicate is true of any object 0, such that substituting 0 for x in the expression results in a true proposition. Like any other predicate, you can construct a proposition from a lambda expression and an argument. In the logical form language, the following is a proposition:

```
(( $\hat{U}$  x (KISS1 k1 x (NAME s1 "Sue")))) (NAME j1 "Jack")) )
```

This proposition is true if and only if (NAME j1 "Jack") satisfies the predicate $(\hat{\lambda} x (\text{KISS1 } k1 x (\text{NAME } s1 "Sue")))$, which by definition is true if and only if

```
(KISS1 k1 (NAME j1 "Jack") (NAME s1 "Sue"))
```

is true. We will often say that this last expression was obtained by **applying** the lambda expression $(\hat{\lambda} x (\text{KISS1 } x (\text{NAME } s1 "Sue")))$ to the argument (NAME j1 "Jack"). This operation is called **lambda reduction**.

Given that we have had to introduce new concepts such as lambda expressions in order to establish a close syntactic-semantic coupling, you might

[Allen 1995: Chapter 9 – Linking Syntax and Semantics / 266]

BOX 9.1 The Lambda Calculus and Lambda Reduction

The lambda calculus is a powerful language based on a simple set of primitives. Formulas in the lambda calculus consist of equality assertions of the form

$\langle\text{expression}\rangle = \langle\text{expression}\rangle$

The most crucial axiom in this system for our purposes is

$$((\hat{\lambda} x. Px) a) = P\{x/a\}$$

where Px is an arbitrary formula involving x and $P\{x/a\}$ is the formula where every instance of x is replaced by a . From this axiom, two principal operations can be defined: lambda reduction (moving from left to right across the axiom) and lambda abstraction (moving from right to left across the axiom). In general, lambda reduction is the principal concern to us because it tends to make formulas simpler. In fact, because lambda reduction simply replaces one formula with a simpler one that is equal to it, the operation is not formally necessary at all to account for semantic interpretation. Without using lambda reduction, however, the answers, though correct, would tend to be unreadable.

be tempted to abandon this approach and develop some other method of semantic interpretation. As a grammar becomes larger and handles more complex phenomena, however, the compositional theory becomes more attractive. For instance, using this method, verb phrases can easily be conjoined even when they have different syntactic structures, as in the sentence

Sue laughs and opens the door.

There are two VPs here: *laughs*, which has a semantic interpretation as a unary predicate true of someone who laughs, say (X a (LAUGHS 112 a)); and *opens the door*, a unary predicate true of someone who opens the door, namely

(\hat{U} a (OPENS1 **12** a < THE **d1** DOOR1>))

These two unary predicates can be combined to form a complex unary predicate that is true of someone who both laughs and opens the door, namely,

(\hat{U} a (& (LAUGHS1 **12** a) (OPENS1 **o1** a <THE **d1** DOOR1>)))

This is in exactly the right form for a VP and can be combined with other constituents like any other VP. For instance, it can be applied to a subject NP with logical form (NAME s1 "Sue") to form the meaning of the original sentence.

(& (LAUGHS1 **12** (NAME **s1** "Sue"))
(OPENS1 **o1** (NAME **s1** "Sue") <THE **d1** DOOR1>))

Consider another example. Prepositional phrase modifiers in noun phrases could be handled in many different ways. For instance, we might not have an independent meaning for the phrase *in the store* in the noun phrase *The man in*

[Allen 1995: Chapter 9 – Linking Syntax and Semantics / 267]

the store. Rather, a special mechanism might be used to look for location modifiers in noun phrases and incorporate them into the interpretation. But this mechanism would then not help in interpreting sentences like *The man is in the store* or *The man was thought to be in the store*. If the prepositional phrase has an independent meaning, in this case the unary predicate

$(\hat{U} \circ (\text{IN-LOC1} \circ \langle \text{THE } \mathbf{s1} \text{ STORE1} \rangle))$

then this same interpretation can be used just as easily as a modifier to a noun phrase (adding a new restriction) or as the predicate of a sentence. The logical form of the noun phrase *the man in the store* would be

$\langle \text{THE } \mathbf{m1} \text{ (MAN1 } \mathbf{m1}) \text{ (IN-LOC1 } \mathbf{m1} \text{ } \langle \text{THE } \mathbf{s1} \text{ STORE1} \rangle) \rangle$

while the logical form of the sentence *The man is in the store* would be

$(\text{IN-LOC1 } \langle \text{THE } \mathbf{m1} \text{ MAN1} \rangle \langle \text{THE } \mathbf{s1} \text{ STORE1} \rangle)$

These are just two simple examples. There are many other generalities that also arise if you adopt a compositional approach to semantics.

In general, each major syntactic phrase corresponds to a particular semantic construction. VPs and PPs map to unary predicates (possible complex expressions built out of lambda expressions), sentences map to propositions,

and NPs map to terms. The minor categories map to expressions that define their role in building the major categories. Since every constituent in the same syntactic category maps to the same sort of semantic construct, these can all be treated uniformly. For example, you don't need to know the specific structure of a VP. As long as its meaning is a unary predicate, you can use it to build the meaning of another larger constituent that contains it.

>> [back](#)

9.2 A Simple Grammar and Lexicon with Semantic Interpretation

This section constructs a simple grammar and lexicon to illustrate how the logical form can be computed using the features while parsing. To keep the examples simple, the logical form will be the predicate-argument structure used in the last section rather than the thematic role representation. This will allow all verbs with the same subcategorization structure to be treated identically. Section 9.4 will then discuss methods of generalizing the framework to identify thematic roles.

The main extension needed is to add a SEM feature to each lexical entry and grammatical rule. For example, one rule in the grammar might be

```
(S SEM (?semvp ?semnp)) -> (NP SEM ?semnp) (VP SEM ?semvp)
```

Consider what this rule does given the NP subconstituent with SEM (NAME m1 "Mary") and the VP subconstituent with SEM

```
( $\hat{U}$  a (SEES1 e8 a (NAME j1 "Jack")))
```

[Allen 1995: Chapter 9 – Linking Syntax and Semantics / 268]

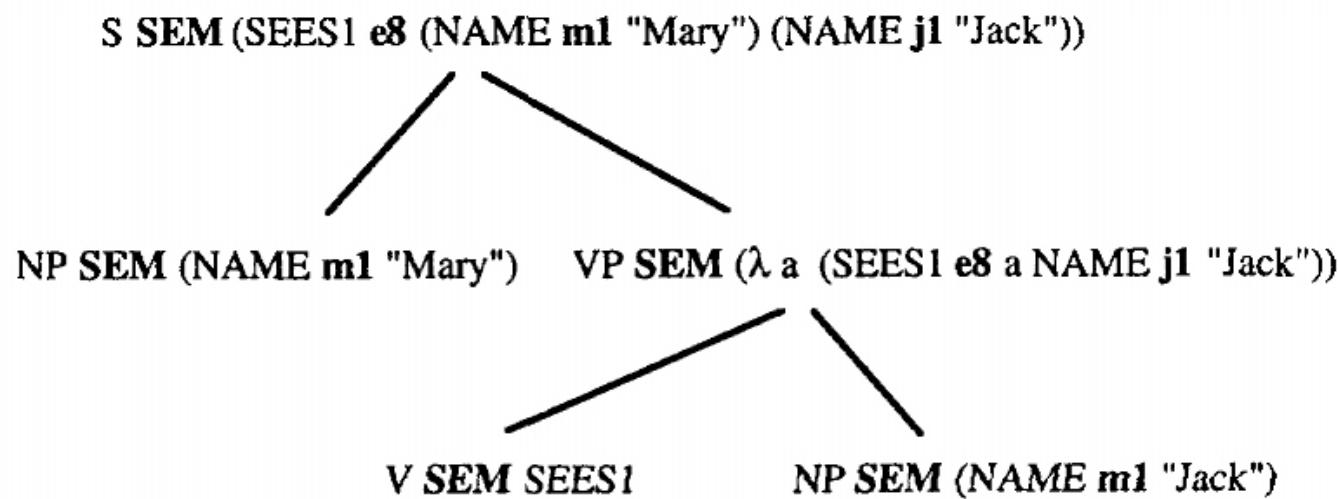


Figure 9.1 A parse tree showing the SEM features

Figure 9.1 A parse tree showing the SEM features

The SEM feature of the new S constituent is simply the expression

```
((  $\hat{U}$  a (SEES1 e8 a (NAME j1 "Jack") ) ) (NAME m1 "Mary") )
```

This expression can be simplified using lambda reduction to the formula

```
(SEES1 e8 (NAME m1 "Mary") (NAME j1 "Jack") )
```

which is the desired logical form for the sentence. Figure 9.1 shows the parse tree for this sentence giving the SEM feature for each constituent.

In the lexicon the SEM feature is used to indicate the possible senses of each word. Generally, a word will have a different word sense for every possible subcategorization it has, since these will be different arity predicates. A sample lexicon is shown in Figure 9.2. When a word has a different SEM form depending on its syntactic features, multiple entries are required. For example, the verb *decide* has two entries: one for the case where the SUBCAT is _none, and one for the case where the SUBCAT is _pp:on, where the verb has an additional argument. Note also that the word *fish* also has two entries because its SEM depends on whether it is singular or plural.

Consider Grammar 9.3, which accepts very simple sentence and verb phrases and computes their logical form. Note that another feature is introduced in addition to the SEM feature. The VAR feature is new and stores the

discourse variable that corresponds to the constituent. It will be useful for handling certain forms of modifiers in the development that follows. The VAR feature is automatically generated by the parser when a lexical constituent is constructed from a word, and then it is passed up the tree by treating VAR as a head feature. It guarantees that the discourse variables are always unique.

The lexical rules for morphological derivation must also be modified to handle the SEM feature. For instance, the rule that converts a singular noun into ~ plural noun takes the SEM of the singular noun and adds the PLUR operators:

```
(N AGR 3p SEM (PLUR ?semn)) ->  
(N AGR 3s IRREG-PL - SEM ?semn) +S
```

[Allen 1995: Chapter 9 – Linking Syntax and Semantics / 269]

```
a (art AGR 3sSEM INDEF1)
```

a (art AGR 3sSEM INDEF1)

can (aux SUBCAT base SEM CAN1)

car (n SEM CAR1 AGR 3s)

cry (v SEM CRY1 VFORM base SUBCAT_none)

decide (v SEM DECIDES1 VFORM base SUBCAT none)

decide (v SEM DECIDES-ON1 VFORM base SUBCAT _pp:on)

dog (n SEM DOG1 AGR 3s)

fish (n SEM FISH1 AGR 3s)

fish (n SEM (PLUR FISH1) AGR 3p)

house (n SEM HOUSE1 AGR 3s)

has (aux VFORM pres AGR 3s SUBCAT pastprt SEM PERF)

a (art AGR 3sSEM INDEF1)

he (pro SEM HE1 AGR 3s)

in (p PFORM {LOC MOT} SEM IN-LOC1)

Jill (name AGR 3s SEM "Jill")

man (n SEM MAN1 AGR 3s)

men (n SEM (PLUR MAN1) AGR 3p)

on (p PFORM LOC SEM ON-LOC1)

saw (v SEM SEES1 VFORM past SUBCAT _np AGR ?a)

see (v SEM SEES1 VFORM base SUBCAT _np IRREG-PAST + EN-PASTPRT +)

she (pro AGR 3s SEM SHE1)

the (art SEM THE AGR {3s 3p})

a (art AGR 3sSEM INDEF1)

to (to AGR - VFORM inf)

Figure 9.2 A small lexicon showing the SEM features

1. (S SEM (?semvp ?semnp) -> (NP SEM ?semnp) (VP SEM ?semvp))
2. (VP VAR ?v SEM ($\hat{\cup}$ a2 (?semv ?v a2))) -> (V[_none] SEM ?semv)
3. (VP VAR ?v SEM ($\hat{\cup}$ a3 (?semv ?v a3 ?semnp))) -> (V[_np] SEM ?semv) (NP SEM ?semnp)
4. (NP WH - VAR ?v SEM (PRO ?v ?sempro)) -> (PRO SEM ?sempro)
5. (NP VAR ?v SEM (NAME 9v ?semname)) -> (NAME SEM ?semname)
6. (NP VAR ?v SEM <?semart ?v (?semcnp ?v)>) -> (ART SEM ?semart) (CNP SEM ?semcnp)

1. (S SEM (?semvp ?semnp) -> (NP SEM ?semnp) (VP SEM ?semvp)

7. (CNP SEM ?semn) -> (NSEM ?semn)

Head features for S, VP, NP, CNP: VAR

Grammar 9.3 A simple grammar with SEM features

A similar technique is used to insert unscoped tense operators for the past and present tenses. The revised morphological rules are shown in Grammar 9.4.

[Allen 1995: Chapter 9 – Linking Syntax and Semantics / 270]

L1. (V VFFORM pres AGR 3s SEM <PRES ?semv>) ->

(V VFFORM base IRREG-PRES - SEM ?semv) +S

L2. (V VFFORM pres AGR {1s 2s 1p 2p 3p} SEM <PRES ?semv>) ->

(V VFFORM base IRREG-PRES - SEM ?semv)

L3. (V VFFORM past AGR {1s 2s 3s 1p 2p 3p} SEM <PAST ?semv>) ->

(V VFFORM base IRREG-PAST - SEM ?semv) +ED

L4. (V VFFORM pastprt SEM ?semv) ->

(V VFFORM base EN-PASTPRT - SEM ?semv) +ED

L5. (V VFFORM pastprt SEM ?semv) ->

(V VFFORM base EN-PASTPRT + SEM ?semv) +EN

L6. (V VFFORM ing SEM ?semv) ->

(V VFFORM base SEM ?semv) +ING

```
L1. (V VFORM pres AGR 3s SEM <PRES ?semv>) ->  
(V VFORM base IRREG-PRES - SEM ?semv) +S
```

```
L7. (N AGR 3p SEM (PLUR ?semn)) ->  
(V AGR 3s IRREG-PL - SEM ?semn) +S
```

Grammar 9.4 The morphological rules with semantic interpretation

These are the same as the original rules given as Grammar 4.5 except for the addition of the SEM feature.

Rule 1 was previously discussed. Rules 2 and 3 handle transitive and intransitive verbs and build the appropriate VP interpretation. Each takes the SEM of the verb, ?semv, and constructs a unary predicate that will apply to the subject. The arguments to the verb sense include an event variable, which is stored in the VAR feature, the subject, and then any additional arguments for subcategorized constituents. Rule 4 constructs the appropriate SEM structure for pronouns given a pronoun sense ?sempro, and rule 5 does the same for proper names. Rule 6 defines an expression that involves an unscoped quantified expression, which consists of the quantifier ?semart, the discourse variable ?v, and a proposition restricting the quantifier, which is constructed by applying the unary predicate ?semcnp to the discourse. variable. For example, assuming that the discourse variable ?v is ml, the NP

the man would combine the SEM of the, namely the operator THE, with the SEM of man, namely MAN1, to form the expression <THE m1 (MAN1 m1)>. Rule 7 builds a simple CNP out of a single N. Since the SEM of a common noun is a unary predicate already, this value simply is used as the SEM of the CNP.

Only two simple modifications need to be made to the standard chart parser to handle semantic interpretation:

- When a lexical rule is instantiated for use, the VAR feature is set to a new discourse variable.
- Whenever a constituent is built, the SEM is simplified by performing any lambda reductions that are possible.

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 271]

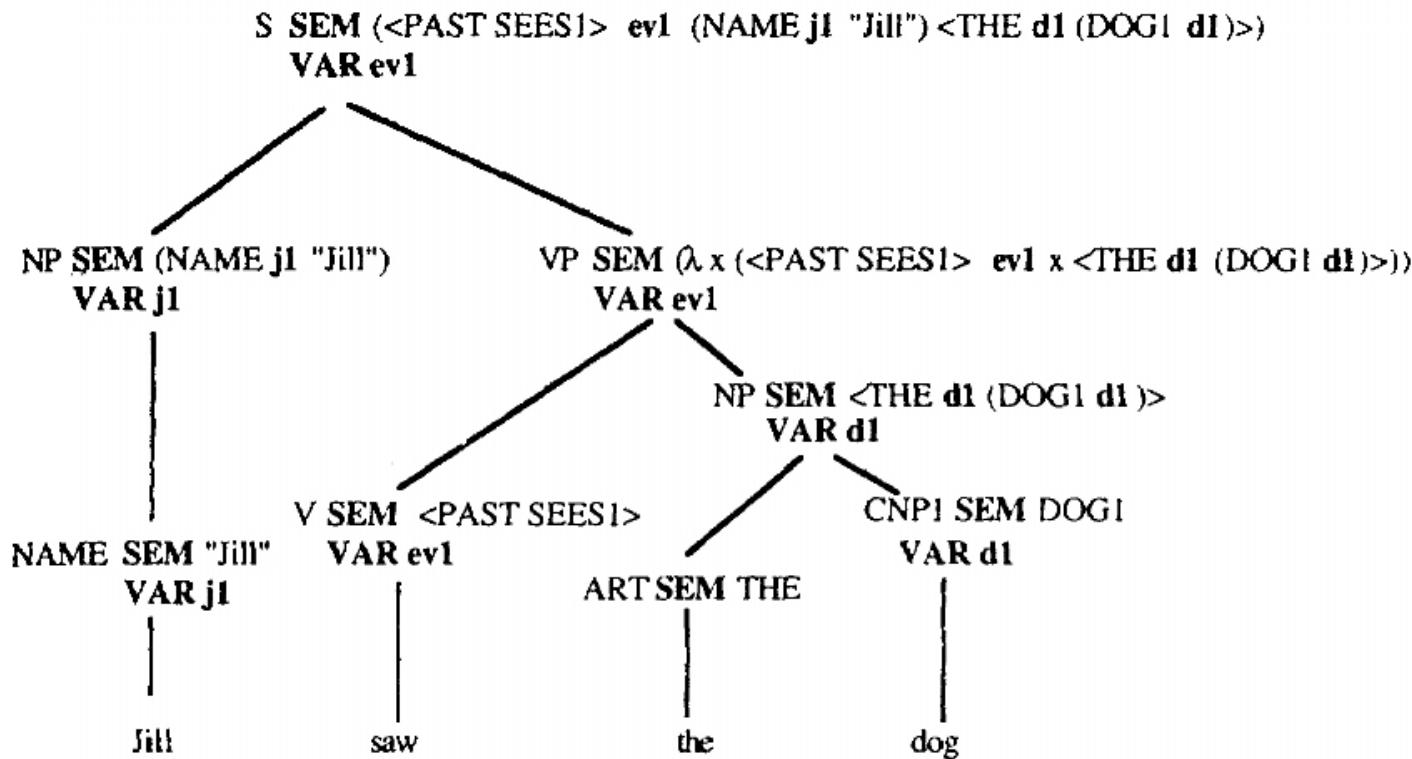


Figure 9.5 The parse of *Jill saw the dog* showing the SEM and VAR features

Figure 9.5 The parse of "Jill saw the dog" showing the SEM and VAR features

With these two changes, the existing parser now will compute the logical form as it parses. Consider a trace of the parser on Jill saw the dog, whose parse tree is shown in Figure.9.5. The word Jill is parsed as a name using the lexical lookup. A new discourse variable, j1, is generated and set to the VAR feature. This constituent is used by rule 5 to build an NP. Since VAR is a head feature, the VAR feature of the NP will be j1 as well, and the SEM is constructed from the equation in the obvious manner. The lexical entry for the word saw generates a V constituent with the SEM <PAST SEES1> and a new VAR **ev1**. Rule 6 combines the SEMs of the two entries for the and dog with the VAR from the noun to build an NP with the SEM <THE d1 (DOG d1)>. This is then combined with the SEM of the verb and its VAR by rule 3 to form a VP with the SEM

(**Û** x (<PAST SEES1> **ev1** x <THE **d1** (DOG1 **d1**)>))

This is then combined with the subject NP to form the final logical form for the sentence.

This completes the description of the basic semantic interpretation process. With the two new features and two minor extensions to the parser described here, a grammar can be specified that builds the logical form as it parses. This technique will work with any of the chart-based parsing strategies that have been discussed in this book.

>> [back](#)

9.3 Prepositional Phrases and Verb Phrases

While the last section introduced everything needed to semantically interpret sentences, only the simplest interpretation techniques were introduced. This

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 272]

section describes some additional examples of grammatical rules that handle slightly more complicated phenomena. Specifically, it addresses the interpretation of verb phrases and prepositional phrases in more detail. First, consider the rule that is needed for handling auxiliary verbs:

```
(VP SEM(Ù a1 (?semaux (?semvp a1)))) ->  
(AUX SUBCAT ?v SEM ?semaux)
```

(VP **VFORM** ?v **SEM** ?semvp)

This rule inserts a modal operator in the appropriate place for the new VP. The SEM equation for the new VP is a little complicated, so consider it in more detail. If ?semaux is a modal operator such as CAN1, and ?semvp is a lambda expression such as $(\hat{U} x (\text{LAUGHS } 1 \ e3 \ x))$, then, according to the auxiliary rule, the SEM of the VP can laugh will be

$(\hat{U} \ a1 \ (\text{CAN1 } ((\hat{U} \ x \ (\text{LAUGHS1 } \mathbf{e3} \ x)) \ a1)))$

This can be simplified to

$(\hat{U} \ a1 \ (\text{CAN1 } (\text{LAUGHS1 } \mathbf{e3} \ a1)))$

It might help to view this type of SEM equation as "lifting" the variable for the subject over the CAN1 operator. Starting with the VP interpretation $(\hat{U} x (\text{LAUGHS1 } e3 \ x))$, the equation builds a new formula containing the CAN1 operator yet still retaining the lambda variable for the subject on the outside of the formula. Note that, like all VPs, the new SEM is a unary predicate that applies to the subject, so the auxiliary rule could be used recursively to analyze more complex sequences of auxiliary verbs.

To analyze prepositional phrases, it is important to realize that they play two different semantic roles in sentences. In one analysis the PP is a modifier to a noun phrase or verb phrase. In the other use the PP is subcategorized for by a head word, and the preposition acts more as a flag for an argument position than as an independent predicate.

Consider the modification case first. In these cases the SEM of the PP is a unary predicate to be applied to whatever it eventually modifies. Thus the following rule would be appropriate for building a PP modifier:

(PP **SEM** (\hat{U} y (?semp y ?semnp))) ->

(P **SEM** ?semp) (NP **SEM** ?semnp)

Given the PP in the corner, if the SEM of the P is IN-LOC1, and the SEM of the NP is <THE c1 (CORNER1 c1>, then the SEM of the PP would be the unary predicate

(\hat{U} y (IN-LOC1 y <THE **c1** CORNER1>))

Now you can consider the interpretation of the noun phrase the tnan in the corner. A reasonable rule that incorporates the PP modifier would be

(CNP **SEM** (\hat{U} n1 (& (?semcnp n1) (?sempp n1)))) ->

(CNP **SEM** ?semcnp) (PP **SEM** ?sempp)

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 273]

Given that the SEM of the CNP man is the unary predicate MAN1 and the SEM of the PP in the corner is (X y (IN! y <THE ci CORNER!>)), the new SEM of the CNP, before simplification, would be

$$(\hat{\cup} \text{ n1 } (\& \text{ (MAN1 n1) } ((\hat{\cup} \text{ y } (\text{IN1 y } <\text{THE c1 CORNER1}>)) \text{ n1})))$$

The subexpression $((\hat{\cup} \text{ y } (\text{IN1 y } <\text{THE c1 CORNER1}>)) \text{ n1})$ can then be simplified to $(\text{IN1 n1 } <\text{THE c1 CORNER1}>)$. Thus the overall expression becomes

$$(\hat{\cup} \text{ n1 } (\& \text{ (MAN1 n1) } (\text{IN1 n1 } <\text{THE c1 CORNER1}>)))$$

This is a unary predicate true of any man who is in the corner, the desired interpretation. Combining this with a quantifier such as the using rule 6 would form a SEM such as

$$<\text{THE m2 } ((\hat{\cup} \text{ z } (\& \text{ (MAN1 z) } (\text{IN1 z } <\text{THE c1 CORNER1}>))) \text{ m2})>$$

which itself simplifies to

$$<\text{THE m2 } (\& \text{ (MAN1 m2) } (\text{IN1 m2 } <\text{THE c1 CORNER1}>))>$$

PPs can also modify verb phrases, as in the VP "cry in the corner" in "Jill can cry in the corner". The syntactic rule that introduces the PP modifier would be

$$\text{VP } \rightarrow \text{ VP PP}$$

You might think that the SEM feature equation can be treated in exactly a parallel way as with the noun phrase case, but there is a complication. Consider the desired behavior. The VP subconstituent cry has the logical form

$(\hat{U} \ x \ (\text{CRIES1 } e1 \ x))$

and the PP has the logical form shown above. The desired logical form for the entire VP cry in the corner is

$(\hat{U} \ a \ (\& \ (\text{CRIES1 } e1 \ a) \ (\text{IN-LOC1 } e1 \ <\text{THE } c1 \ \text{CORNER1}>)))$

The problem is that the unary predicate for the VP subconstituent is intended to apply to the subject, where the unary predicate for the PP modifier must apply to the discourse variable generated for the VP. So the technique used for rule 9 does not yield the correct answer. Rather, the SEM constructed from the PP must be applied to the discourse variable instead. In other words, the appropriate rule is

$(\text{VP } \mathbf{VAR} \ ?v \ \mathbf{SEM} \ (\hat{U} \ x \ (\& \ (?semvp \ x) \ (?sempp \ ?v)))) \rightarrow$
 $(\text{VP } \mathbf{VAR} \ ?v \ \mathbf{SEM} \ ?semvp) \ (\text{PP } \mathbf{SEM} \ ?sempp)$

The parse tree for the VP cry in the corner using this rule is shown in Figure 9.6.

Prepositional phrases also appear as subcategorized constituents in verb phrases, and these cases must be treated differently. Specifically, it is the verb that determines how the prepositional phrase is interpreted. For example, the PP on a couch in isolation might indicate the location of some object or event, but with the verb decide, it can indicate the object that is being decided about. The

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 274]

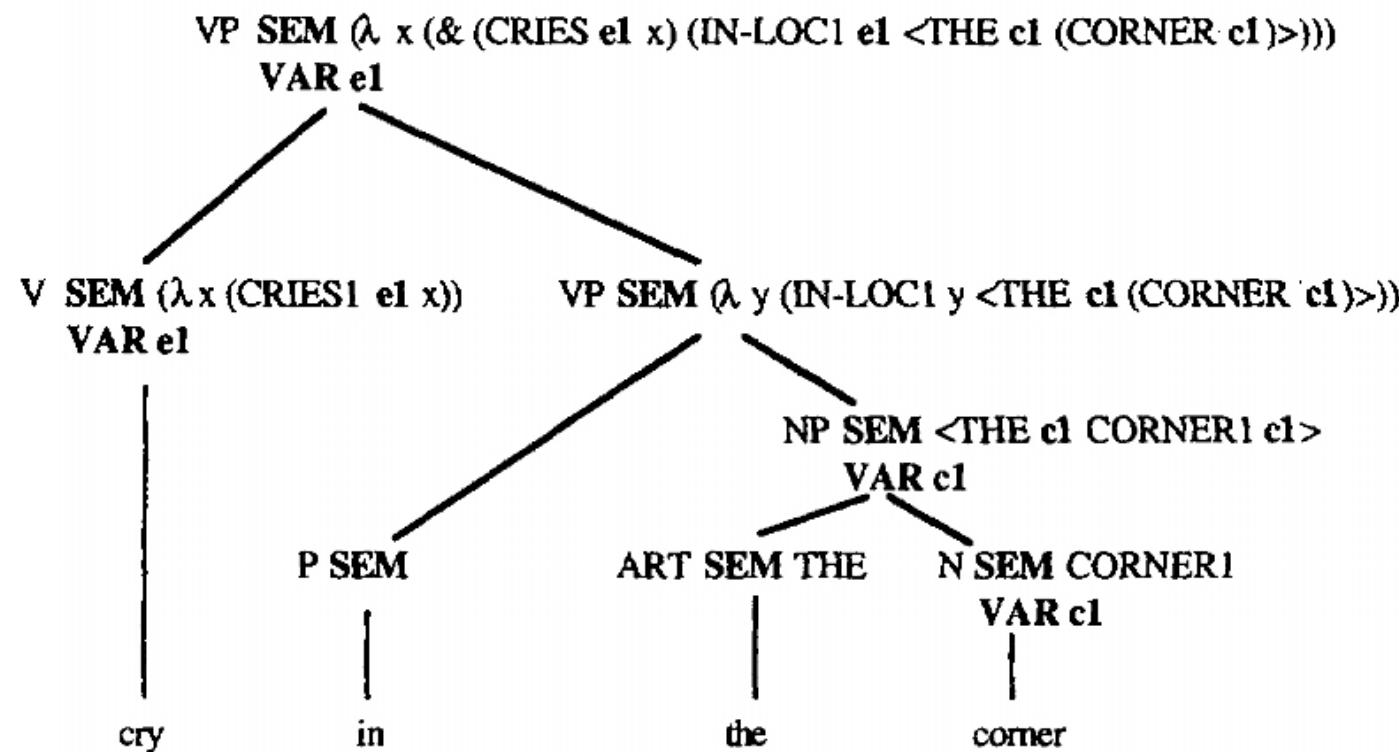


Figure 9.6 Using the VAR feature for PP modifiers of VPs

Figure 9.6 Using the VAR feature for PP modifiers of VPs

distinction between the two readings can be illustrated by considering that the sentence Jill decided on a couch is ambiguous between two readings:

- Jill made a decision while she was on a couch.
- Jill made a decision about a couch.

The first case results from treating on a couch as an adverbial PP. which was discussed above. What is the semantic interpretation equation for the second case? The appropriate syntactic rule is

VP → V[_pp:on] NP PP[on]

and the desired logical form of the final VP is

(\hat{U} s (DECIDES-ON1 **d1** s <A **c1** (COUCH **c1**)>))

Note that there appears to be no semantic contribution from the word on in this case. Subcategorized PPs are handled in many systems by distinguishing between two different types of prepositional phrases. A new binary feature PRED is introduced, where + indicates that the prepositional phrase should be interpreted as a predicate, and — means it should be interpreted as an argument, that is, a term. This binary distinction allows us to specify two rules for prepositional phrases. These are rules 8 and 9 in Grammar 9.7. Rule 8 is restricted to prepositional phrases with a +PRED value, and the PP acts as a modifier. Rule 9 handles all the subcategorized prepositional

phrases, with the -PRED feature, in which case the SEM of the PP is simply the SEM of the object NP. Grammar 9.7 also summarizes all the other rules developed in this section.

Figure 9.8 shows the two readings of the VP decide on a couch given the grammar defined by Grammars 9.3 and 9.7. The case in which the decision is

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 275]

8. $(PP \text{ PRED} + SEM \ (\hat{\cup} \ x \ (?semp \ x \ ?semnp))) \rightarrow$
 $(P \text{ SEM } ?semp) \ (NP \text{ SEM } ?semnp)$
9. $(PP \text{ PREI}) \ - \ PFORM \ ?pf \ SEM \ ?semnp \rightarrow$
 $(P \text{ ROOT } ?pf) \ (NP \text{ SEM } ?semnp)$
10. $(VP \text{ VAR } ?v \ SEM \ (\hat{\cup} \ ag1 \ (\& \ (?semvp \ ag1) \ (?sempp \ ?v)))) \rightarrow$
 $(VP \text{ SEM } ?semvp) \ (PP \text{ PRED} + SEM \ ?sempp)$

8. (PP PRED + SEM (\hat{U} x (?semp x ?semnp))) \rightarrow

(P SEM ?semp) (NP SEM ?semnp)

11. (VP VAR ?v SEM (\hat{U} ag2 (?semv ?v ag2 ?sempp))) \rightarrow

(V[_np_pp:on SEM ?semv] (PP PRED - PFORM on SEM ?sempp))

12. (VP SEM (\hat{U} a1 (?semaux (?semvp a1)))) \rightarrow

(AUX SUBCAT ?v SEM ?semaux) (VP VFORM ?v SEM ?semvp)

13. (CNP SEM (A. n1 (& (?semcnp n1) (?sempp n1)))) \rightarrow

(CNP SEM ?semcnp) (PP PRED + SEM ?sempp)

Head features for PP: PFORM

Head features for VP, CNP: VAR

Grammar 9.7 Rules to handle PPs in verb phrases

about a couch is shown in the upper half of the figure: the PP has the feature

-PRED, as previously discussed, and its SEM is <A c1 COUCH1>. The case in which a decision is made on a couch is shown in the lower half of the figure: the PP has the feature +PRED and its SEM is (\hat{U} x (ON-LOC1 x <A c1 COUCH1>)).

>> [back](#)

9.4 Lexicalized Semantic Interpretation and Semantic Roles

So far, the semantic forms of the lexical entries have only consisted of the possible senses for each word, and all the complexity of the semantic interpretation is encoded in the grammatical rules. While this is a reasonable strategy, many researchers use a different approach, in which the lexical entries encode the complexities and the grammatical rules are simpler. Consider the verb decide, say in its intransitive sense, DECIDES1. The SEM of the verb is simply the sense DECIDES1, and rule 2 builds the lambda expression $(\lambda y (\text{DECIDES1 } e1 \ y))$. An alternative approach would be to define the SEM of the lexical entry to be $(\lambda y (\text{DECIDES1 } e1 \ y))$, and then the SEM equation for rule 2 would just use the SEM of the verb as its SEM. Likewise, the SEM for the lexical entry for the transitive sense could be the expression $(\lambda o (\lambda y (\text{DECIDES-ON1 } e1 \ y \ o)))$. The SEM equation for rule 3 would then apply this predicate to the SEM of the object to obtain the appropriate SEM, as before.

There is a tradeoff here between the complexity of the grammatical rules and the complexity of the lexical entries. With the grammar we have developed so far, it seems better to stay with the simple lexicon. But if the logical forms become more complex, the alternative approach begins to look attractive. As an

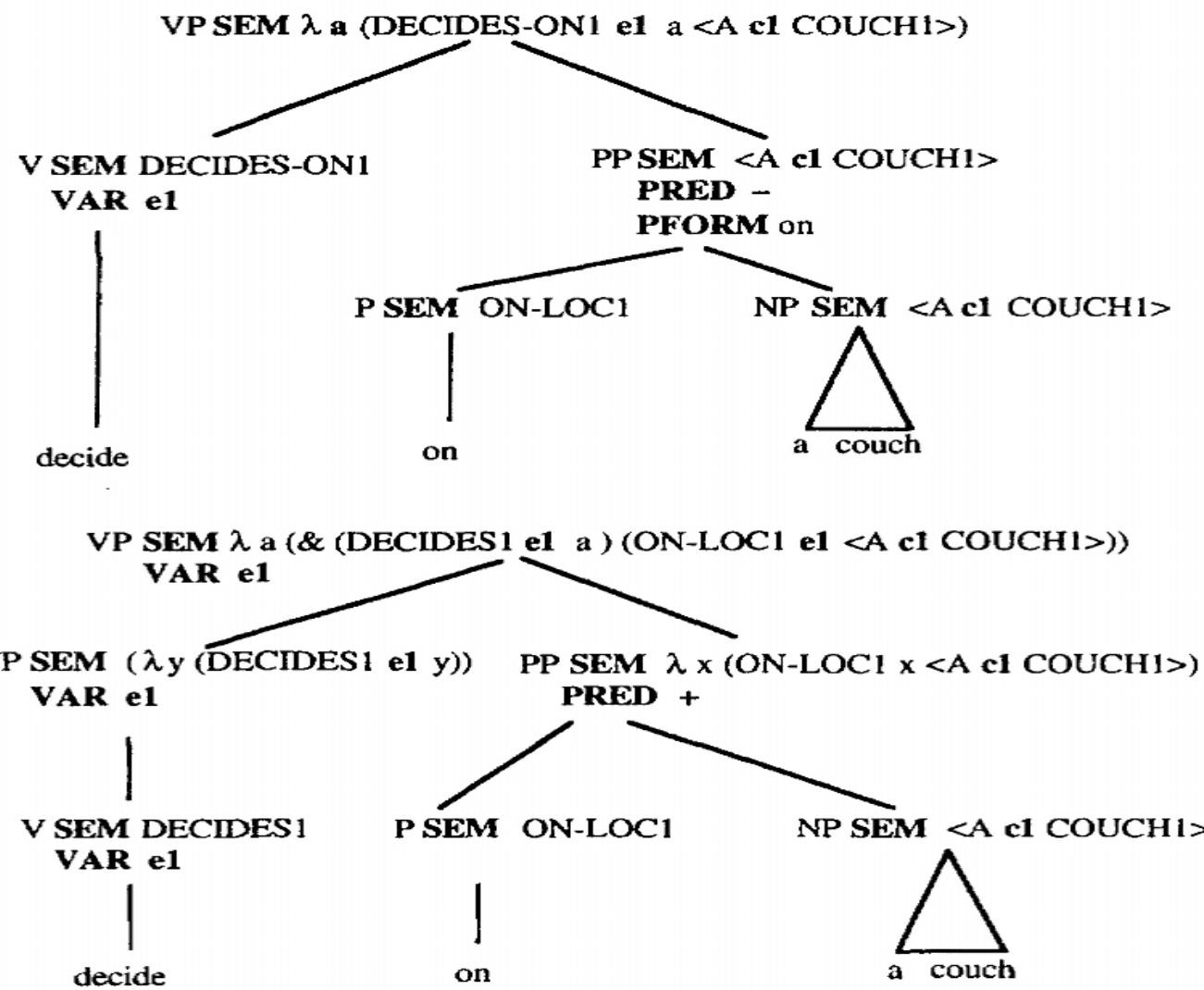


Figure 9.8 Two possible parse trees for the VP *decide on a couch*

Figure 9.8 Two possible parse trees for the VP "decide on a couch"

example, consider how to specify a grammar that produces a logical form based on thematic roles. Consider first what happens if the lexicon can only store atomic word senses. Whereas the earlier grammar had only one rule that could cover all transitive verbs, the new grammar would have to classify transitive verbs by the thematic roles they use, and a separate rule must be used for each. The verbs see and eat, for example, both have transitive forms where the subject fills the AGENT role and the object fills the THEME role. The verb break, on the other hand, also has a sense where the subject fills the INSTR role and the object fills the THEME role, as in The hammer broke the window. To handle these two cases, a new feature, say ROLES, would have to be added to the lexical entries that identifies the appropriate forms, and then a separate grammar rule added for each. These might look as follows:

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 277]

(VP **VAR** ?v **SEM** ($\hat{\cup}$ a (?semv ?v [AGENT a] [THEME ?semnp])) ->

(V **ROLES** AG-THEME **SEM** ?semv) (NP **SEM** ?semnp)

(VP **VAR** ?v **SEM** ($\hat{\cup}$ a (?semv ?v [INSTR a] [THEME ?semnp])) ->

(V **ROLES** INSTR-THEME **SEM** ?semv) (NP **SEM** ?semnp)

Additional rules would be added for all the other possible combinations of roles that can be used by the verbs.

Clearly, this approach could be cumbersome. Since it requires adding thematic role information to the lexicon anyway (using the ROLES feature), it might be simpler just to encode the appropriate forms in the lexicon. For instance, if the lexical entries are

see: (V **VAR** ?v **SEM** ($\hat{\cup}$ o ($\hat{\cup}$ a (SEES1 ?v [AGENT a] [THEME ?o]))))

break: (V **VAR** ?v **SEM** ($\hat{\cup}$ o ($\hat{\cup}$ a (BREAKS1 ?v [INSTR a]

[THEME ?o]))))

then a single grammar rule, namely,

(VP **SEM** (?semv ?semnp)) ->

(V SEM ?semv) (NP SEM ?semnp)

will cover all the cases. Consider the VP see the book, where the SEM of "see" is as above and "the book" is <THE b1 (BOOK1 b1)>. The SEM for the VP would be

(($\hat{U} \circ (\hat{U} a (SEES1 b1 [AGENT a] [THEME 0]))$) <THE b1 (BOOK1 b1)>)

which can be simplified using one lambda reduction-to

($\hat{U} a (SEES1 b1 [AGENT a] [THEME <THE b1 (BOOK1 b1)>])$)

The same rule given the VP break the book would apply the SEM of break above to the SEM for the book to produce the reduced logical form

($\hat{U} a (BREAKS1 b1 [INSTR a] [THEME <THE b1 (BOOK1 b1)>])$)

>> [back](#)

Hierarchical Lexicons

The problem with making the lexicon more complex is that there are many words, and specifying a lexicon is difficult even when the entries are simple. Just specifying the semantic interpretation rules for the most common sense is tedious, as there is a different semantic interpretation rule for each complement structure the verb allows. For example, the verb give allows the forum

I gave the money.

I gave John the money.

I gave the money to John.

The lexical entries for give would include the following:

```
(V SUBCAT _np  
    SEM Ü o Ü a (GIVE1 * [AGENT a] [THEME o]))
```

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 278]

```
(V SUBCAT _np_np
```

```

SEM Ù r Ù o Ù a (GIVE1 * [AGENT a] [THEME o] [TO-POSS r] )

(V SUBCAT _np_pp:to

SEM Ù o Ù r Ù a (GIVE1 * [AGENT a] [THEME o] [TO-POSS r] )

```

This is quite a burden if it must be repeated for every verb. Luckily, we can do much better by exploiting some general regularities across verbs in English. For example, there is a very large class of verbs, including most transitive verbs, that all use the same semantic interpretation rule for the ..np SUBCAT form. This class includes verbs such as give, take, see, find; paint, and so on - virtually all verbs that describe actions. The idea of a hierarchical lexicon is to organize verb senses in such a way that their shared properties can be captured concisely. This depends on a technique called inheritance, where word senses inherit or acquire the properties of the abstract classes above them in the hierarchy. For example, a very useful lexical hierarchy can be based on the SUBCAT and SEM properties of verbs. Near the top of the hierarchy are abstract verb senses that define common verb classes. For example, the abstract class INTRANS-ACT defines a class of verbs that allow a SUBCAT _none and have the semantic interpretation rule

```

Ù s (?PREDN * [AGENT s] )

```

where ?PREDN is a predicate name determined by the verb. This fully specifies the semantic interpretation of intransitive verbs such as *run*, *laugh*, *sit*, and so on, except for the actual predicate name that still needs to be specified in the lexical entry for the word. Another common form is the simple transitive verb that describes an action, including the verbs listed above. This form, TRANS-ACT, has a SUBCAT _np and a SEM Ù o Ù a (? PREDN *[AGENT a] [THEME o]).

We can define similar classes for all the common verb forms and then build an inheritance structure that relates verb senses to the forms they allow. Figure 9.9 shows a lexical hierarchy that encodes the definitions of four different verb senses. It is equivalent to the following entries specified without a hierarchy:

run (in the intransitive exercise sense, RUN1):

```
(SUBCAT _none  
SEM  $\hat{U}$  a (RUN1 * [AGENT a]))
```

run (in the transitive "operate" sense, OP1):

```
(SUBCAT _np  
SEM  $\hat{U}$   $\circ$   $\hat{U}$  a (OP1 * [AGENT a] [THEME  $\circ$ D]))
```

donate (allowing the transitive and "to" form):

```
(SUBCAT np  
SEM  $\hat{U}$   $\circ$   $\hat{U}$  a (DONATE1 * [AGENT a] [THEME  $\circ$ ]))  
(SUBCAT np pp:to  
SEM  $\hat{U}$   $\circ$   $\hat{U}$  r  $\hat{U}$  a (DONATE1 * [AGENT a] [THEME  $\circ$ ] [TO-POSS r]))
```

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 279]

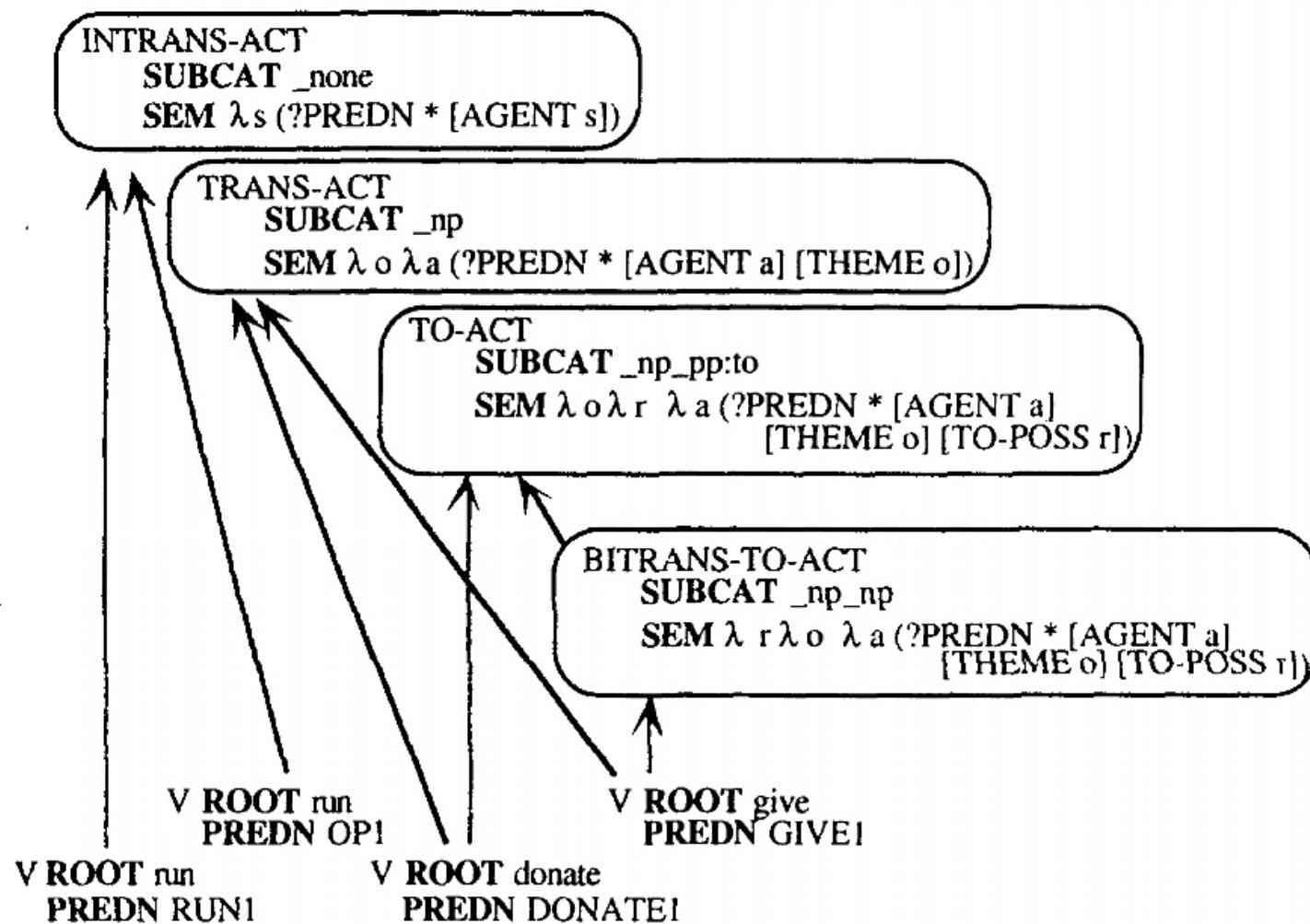


Figure 9.9 Part of a lexical hierarchy for verbs based on SUBCAT and SEM features

Figure 9.9 Part of a lexical hierarchy for verbs based on SUBCAT and SEM features

and, of course,

give (in all its forms as previously discussed):

(**SUBCAT** np

SEM \hat{U} o \hat{U} a (GIVE1 * [AGENT a1 [THEME o]]))

(**SUBCAT** _np_pp:to

SEM \hat{U} o \hat{U} r \hat{U} a (GIVE1 * [AGENT a] [THEME o] [TO-POSS r]))

(SUBCAT _np_np

SEM \hat{U} r \hat{U} o \hat{U} a (GIVE1 * [AGENT a] [THEME o] [TO-POSS r]))

You could implement a lexical hierarchy by adding another feature to each lexical entry called SUP (for superclass), which has as its value a list of abstract categories from which the constituent inherits properties. It is then relatively simple to write a program that searches up this hierarchy to find all the relevant feature values whenever the lexicon is accessed. The entry for the verb *give* might now look like this:

```
give: (V ROOT give  
PREDN GIVE1  
SUP (BITRANS-TO-ACT TRANS-ACT))
```

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 280]

14. (S **INV** - **SEM** (WH-query ?sems)) ->
(NP **WH** Q **AGR** ?a **SEM** ?semnp)
(S **INV** + **SEM** ?sems **GAP** (NP **AGR** ?a **SEM** ?semnp))

15. (S **INV** + **GAP** ?g **SEM** (?semaux (?semvp ?semnp))) ->

(AUX **AGR** ?a **SUBCAT** ?s **SEM** ?semaux)

(NP **AGR** ?a **GAP** - **SEM** ?sempp)

(VP **VFORM** ?s **GAP** ?g **SEM** ?semvp)

16. (NP **WH** ? **VAR** ?v **SEM** <WH ?v (?sempro ?v)>) ->

(PRO **WH** ? **SEM** ?sempro)

Grammar 9.10 Rules to handle simple wh-questions

>> [back](#)

9.5 Handling Simple Questions

The grammar developed so far deals only with simple declarative sentences. To extend the grammar to handle other sentence types requires adding rules to interpret wh-terms, inverted sentences, and the gap propagation required to handle wh-questions. This can be done quite directly with the mechanisms developed so far and requires no additional extensions to the parser. All you need to do is augment the S rules first developed for questions in Chapter 5 with appropriate SEM feature equations. .

The part that may need some explanation is how the SEM feature interacts with the GAP feature. As you recall, many wh-questions use the GAP feature to build the appropriate structures. Examples are

Who did Jill see?

Who did Jill want to see?

Who did Jill want to win the prize?

The rule introduced in Chapter 5 to account for these questions was

(S **INV** -) \rightarrow (NP **WH** Q **AGR** ?a) (S **INV** + **GAP**(NP **AGR** ?a))

That is, a wh-question consists of an NP with the WH feature Q followed by an inverted sentence with an NP missing that agrees with the first NP. To make the semantic interpretation work, we add the SEM feature to the features of the gap. This way it is passed into the S structure and can be used at the appropriate place when the gap is found. The revised rule is

```
(S INV - SEM (WH-query ?sems) ) ->  
(NP WH Q AGR ?a SEM ?semnp)  
(S INV + SEM ?sems GAP (NP AGR ?a SEM ?semnp) )
```

Grammar 9.10 gives the new rules required to handle this type of question. Rule 14 was just discussed. Rule 15 handles inverted sentences and could be used

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 281]

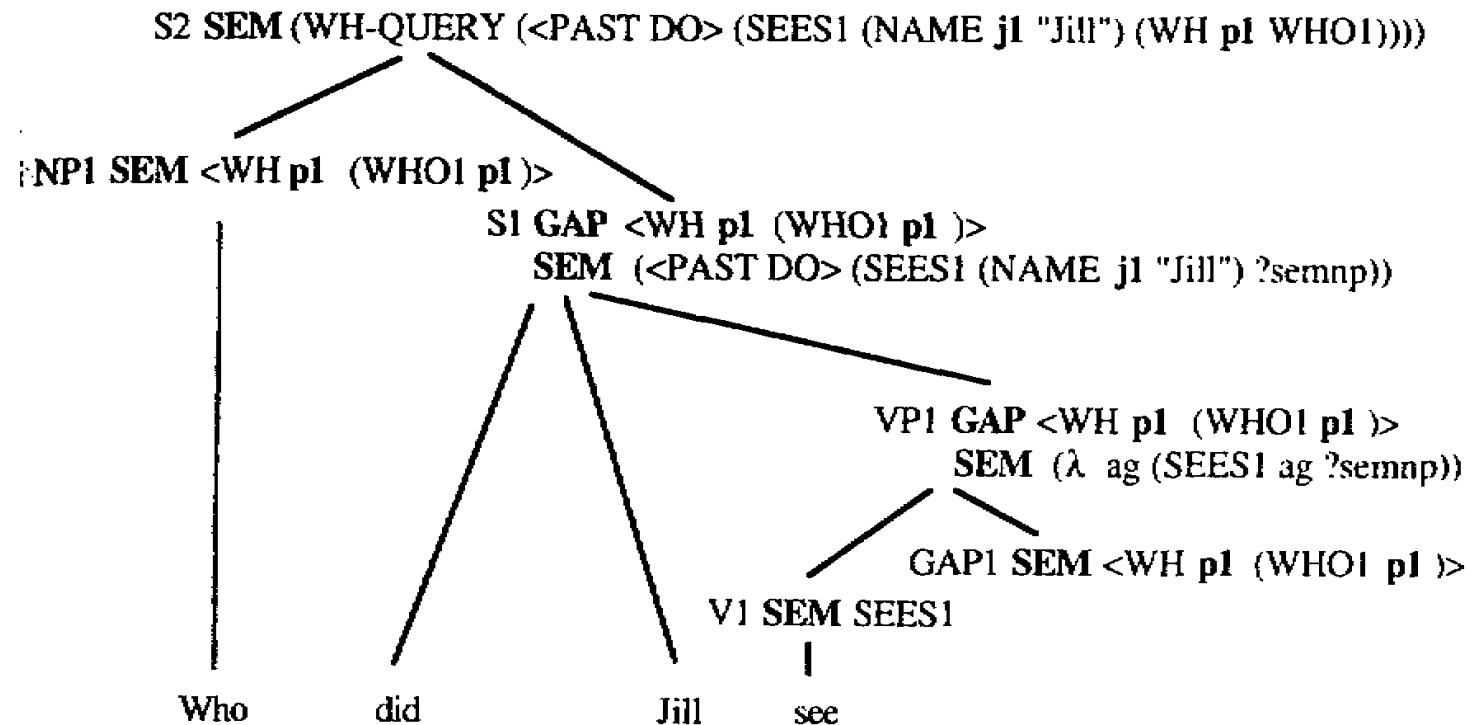


Figure 9.11 The parse tree for *Who did Jill see?*

Figure 9.11 The parse tree for "Who did Jill see?"

for yes/no questions as well as for the wh-questions discussed here. Rule 17 allows noun phrases to be built from wh-pronouns, such as *who* and *what*.

The lexical entry for the wh-words would have to be extended with a SEM feature as well. For instance, the word *who* would have the entry

```
(PRO WH {Q R} SEM WHO1 AGR {3s 3p})
```

The predicate WHO1 would be true of any objects that are reasonable answers to such questions, including people and possibly other animate agents.

To see how the SEM feature and GAP features interact, consider a derivation of the parse tree for the question *Who did Jill see?*, as shown in Figure 9.11. Using rule 16, the word *who* would be parsed as the noun phrase

```
(NP WH Q AGR 3s SEM <WH p1 (WHO1 p1)>)
```

This constituent can be used to start rule 14, and thus we need the following constituent to complete the rule:

```
(S INV +  
GAP (NP AGR 3s SEM <WH p1 (WHO1 p1)>)  
SEM ?sems)
```

Rule 15 can be used to rewrite this, and the words *did* and *Jill* provide the first two subconstituents. The third subconstituent needed is:

```
(VP VFORM base  
  GAP (NP AGR 3s SEM <WH p1 (WHO1 p1)>  
  SEM ?semvp)
```

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 282]

This is a VP with an NP gap. Rule 3 from Grammar 9.3 applies to transitive verbs such as see. The GAP feature is used to fill a gap for the object of the verb, instantiating ?semnp to <WH p1 (WHO p1)>. Thus the SEM of the new VP is

```
(U a3 (SEES1 s1 a3 <WH p1 (WHO1 p1)>))
```

The SEM of the initial word who has found its way to the appropriate place in the sentence. The parse can now be completed as usual. This technique of passing in -the SEM in the GAP is completely general and can be used to handle all of the question types discussed in Chapter 5.

>> [back](#)

o Prepositional Phrase Wh-Questions

Questions can also begin with prepositional phrases, such as In which box did you put the book?, Where did you put the book?, and When did he disappear? The semantic interpretation of these questions will depend on whether the PPs are subcategorized for the verb or are VP modifiers. Many such questions can be handled by a rule virtually identical to rule 14 for NP wh.questions, namely

```
(S INV - SEM (WH-query ?sems)) ->  
  
(PP WH Q PRED ?p PTYPE ?pt SEM ?sempp)  
  
(S INV + SEM ?sems GAP (PP PRED ?p PTYPE ?pt SEM ?sempp))
```

To handle wh-terms like where appropriately, the following rule is also needed:

```
(PP PRED ?pd PTYPE ?pt SEM ?sem) ->
```

```
(PP-WRD PRED ?pd PTYPE ?pt SEM ?sem)
```

The wh-term where would have two lexical entries, one for each PRED value:

```
(PP-WRD PTYPE {LOC MOT} PRED - VAR ?v  
SEM <WH ?v (LOC1 ?v)>  
(PP PRED + VAR ?v SEM (U x (AT-LOC x <WH ?v (LOC1 v)>) ))
```

These rules would extend the existing grammar so that many such questions could be answered. Figure 9.12 shows part of the parse tree for the question "Where did Jill go?".

Note that handling questions starting with +PRED prepositional phrases depends on having a solution to the problem concerning gap propagation first mentioned in Chapters. Specifically, the rule VP -+ VP PP, treated in the normal way, would only pass the GAP into the VP subconstituent, namely the nonlexical head. Thus there seems no way to create a PP gap that modifies a verb phrase. But this was a problem with the syntactic grammar as well, and any solution at that level should carry over to the semantic level.

S4 SEM (WH-QUERY (GOES1 g1 (NAME j1 "Jill") <WH II LOC1>))

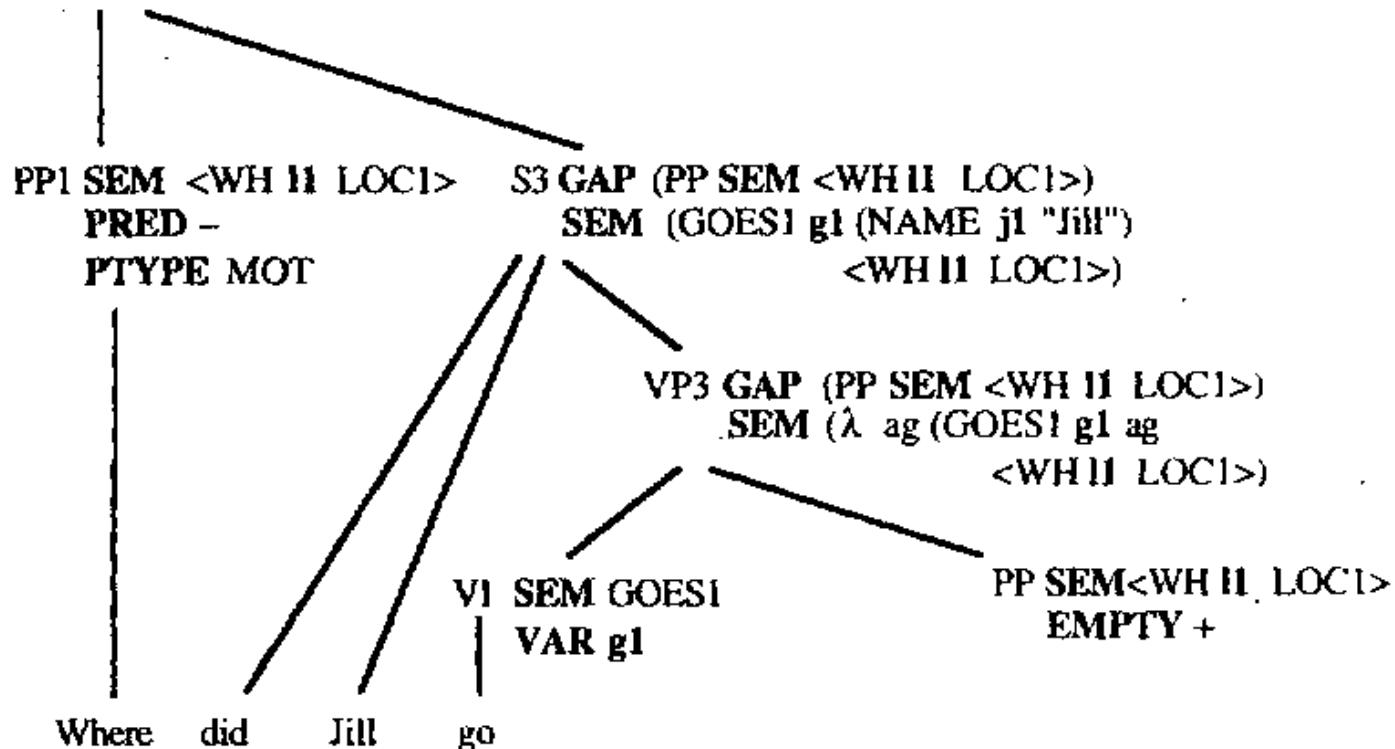


Figure 9.12 The parse tree for *Where did Jill go?*

Figure 9.12 The parse tree for "Where did Jill go?"

>> [back](#)

9.6 Semantic Interpretation Using Feature Unification

So far we have used lambda expressions and lambda reduction to drive the semantic interpretation. This provides a good framework for explaining and corner -paring techniques for semantic interpretation. However, many systems do not explicitly use lambda expressions and perform semantic interpretation directly using feature values and variables. The basic idea is to introduce new features for the argument positions that earlier would have been filled using lambda reduction. For instance, instead of using rule 1 in Grammar 9.3, namely

$$(S \text{ SEM } (?semvp \text{ ?semnp})) \rightarrow (NP \text{ SEM } ?semnp) (VP \text{ SEM } ?sernvp)$$

a new feature **SUBJ** is introduced, and the rule becomes

$$(S \text{ SEM } ?semvp) \rightarrow (NP \text{ SEM } ?semnp) (VP \text{ SUBJ } ?semnp \text{ SEM } ?semvp)$$

The SEM of the subject is passed into the VP constituent as the SUBJ feature and the SEM equations for the VP insert the subject in the correct position. The new version of rule 3 in Grammar 9.3 that does this is

$$\begin{aligned} & (VP \text{ VAR } ?v \text{ SUBJ } ?semsubj \text{ SEM } (?semv \text{ ?v } ?semsubj \text{ ?semnp})) \rightarrow \\ & (V[_{\text{none}}] \text{ SEM } ?semv) (NP \text{ SEM } ?semnp) \end{aligned}$$

Figure 9.13 shows how this rule builds the SEM of the sentence Jill saw the dog. Compare this to the analysis built using Grammar 9.3 shown in Figure 9.5. The differences appear in the treatment of the VP. Here the SEM is the full proposition with the subject inserted, whereas before the SEM was a lambda expression that would be applied to the subject later in the rule that builds the S.

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 284]

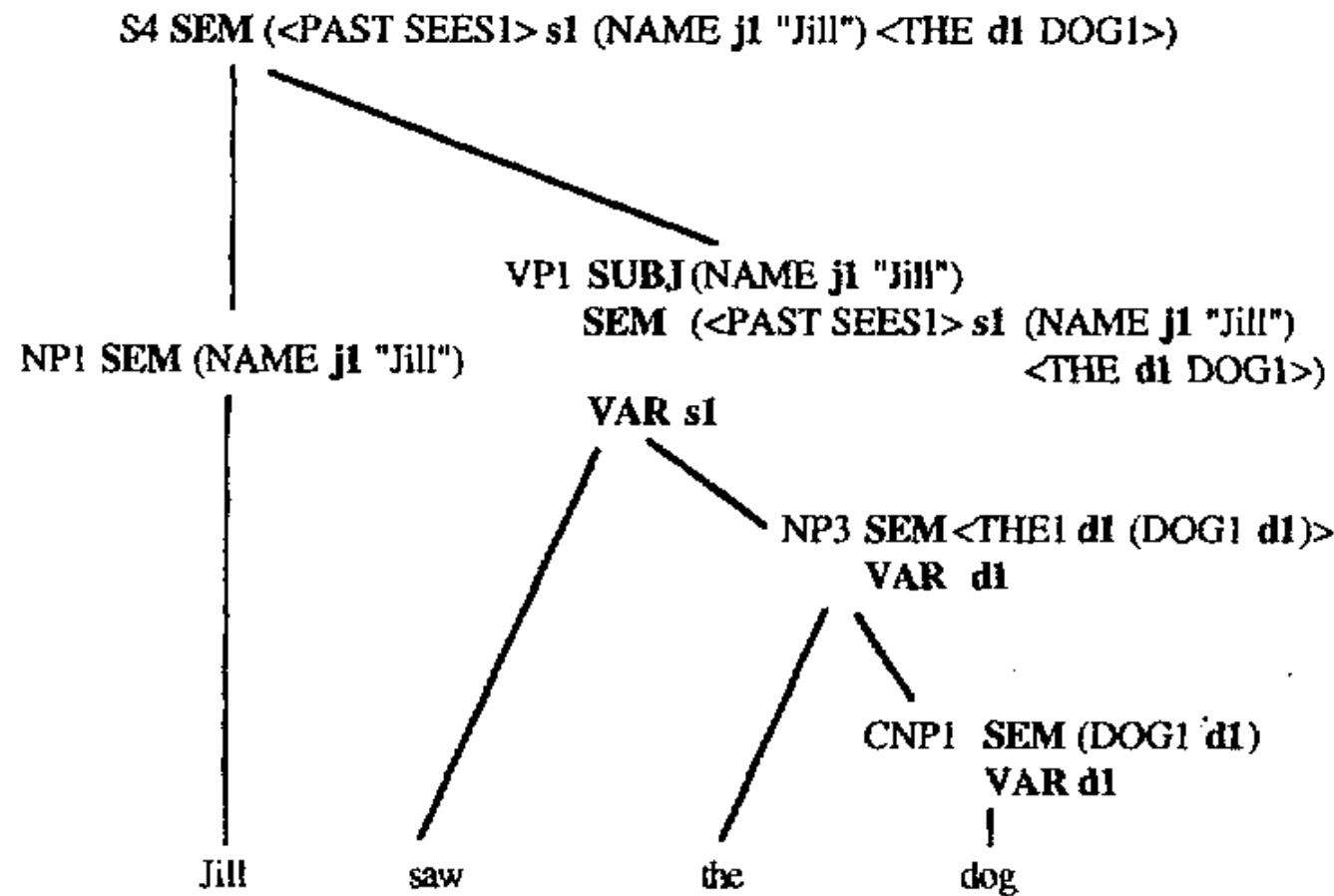


Figure 9.13 The parse tree for *Jill saw the dog* using the SUBJ feature

Figure 9.13 The parse tree for Jill saw the dog using the SUBJ feature

1. (S SEM ?semvp) ->
(NP SEM ?semsubj) (VP SUBJ ?semsubj SEA! ?semvp)
2. (VP VAR ?v SUBJ ?semsubj SEM (?semv ?v ?semsubj)) ->
(V[nonej SEA! ?semv])
3. (VP VAR ?v SUBJ ?semsubj SEM (?semv ?v ?semsubj ?semnp)) ->
(V[_np] SEM ?semv) (NP SEM ?semnp)
4. (NP VAR ?v SEM (PRO ?v ?sempro)) -> (PRO SEM ?sempro)
5. (NP VAR ?v SEM (NAME ?v ?semname)) -> (NAME SEM ?semname)
6. (NP VAR ?v SEM <?semart ?v ?semcnp>) ->
(ART SEM ?semart) (CNP SEM ?semcnp)

7. (CNP VAR ?v SEM (?semn ?v)) -* (N SEM ?semn)

Head features for S, VP, NP, CNP: VAR

Grammar 9.14 A simple grammar with SEM features

Grammar 9.14 is a version of Grammar 9.3 reformulated using this technique. Besides the changes to rules 1, 2, and 3, rules 6 and 7 are also modified. Rule 7 uses the VAR value to build a full proposition (as opposed to a unary predicate in the old grammar), and rule 6 is changed appropriately to account for the change to rule 7.

>> [back](#)

BOX 9.2 Encoding Semantics Entirely in Features

A more radical departure from the approach described in this chapter encodes the entire logical form as a set of features. For instance, rather than the logical form previously developed, the sentence "Jill saw the dog" might have a SEM of

```
(PRED SEES1  
VAR s1  
TNS PAST  
AGENT (NAME j1 "Jill")  
THEME (SPEC THE1  
VAR d1  
RESTRICTION (PRED DOG1
```

ARG1 **d1**)))

Logical forms like this can easily be constructed using additional features for each semantic slot. For instance, here are new versions of the first three rules in Grammar 9.14 that compute this type of logical form:

1. (S **VFORM** past **SEM** ?semvp) ->
(NP SEM ?semnp) (VP TNS past SUBJ ?semnp SEM ?semvp)
2. (VP TNS ?tns **SUBJ** ?semsubj SEM ?semv) ->
(V[_none] **VAR** ?v STJBJ ?semsubj TNS ?tns SEM ?semv)
3. (VP TNS ?tns **SUBJ** ?subj SEM ?semv) ->
(V[np] **VAR** ?v **SUBJ** ?subj **OBJVAL** ?obj **TNS** ?tns **SEM** ?semv) (NP **SEM** ?obj)

Note that all the arguments are passed down into the lexical rule, which would assemble the appropriate semantic form. For instance, the lexical entry for saw might be

(V[_np] **SUBJ** ?s **OBJVAL** ?o **VAR** ?v **TNS** past

```
SEM (PRED SEES1 VAR ?v TNS PAST AGENT ?s THEME ?o) )
```

When this entry is unified with the V in rule 3, the appropriate SEM structure would be built and bound to the variable ?semv, as desired.

One advantage of this approach is that no special mechanism need be introduced to handle semantic interpretation. In particular, there is no need to have a lambda reduction step. Everything is accomplished by feature unification. Another significant advantage is that a grammar specified in this form is reversible and hence can be used to generate sentences as well as parse them, as discussed in the next section. Not all lambda expressions can be eliminated using these techniques, however. For instance, to handle conjoined subject phrases as in Sue and Sam saw Jack, the meaning of the verb phrase must still be a lambda expression. If the subject were inserted into the VP using a variable for the SUBJ feature, then this variable would have to unify with the SEMs of both Sue and Sam, which it can't do.

>> [back](#)

o 9.7 Generating Sentences from Logical Form

Intuitively, once you have a grammar for parsing, it should be easy to reverse it and use it for generation. Such a sentence generator would be given a constituent with its SEM feature set to a logical form, and then it would use the grammar to decompose this constituent into a series of lexical constituents that would have

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 287]

the appropriate meaning. Not all grammars are reversible, however. In fact, Grammar 9.3 is not reversible because it uses lambda reduction. To see why, consider an example. Say you want to generate a sentence that has the meaning

(<PAST SEES1> **s1** (NAME **j1** "Jill") <THE **d1** (DOG1 **d1**)>)

Grammar 9.3 has only one S rule, and if you try to unify the SEM value in rule 1 with this logical form it will fail. The pattern (?semvp ?semnp) would match any proposition built out of a unary predicate and one argument, but the logical form specified is a proposition with three arguments. The problem is that lambda reduction was used to convert the original logical form, which was

$$((\hat{U} \ a \ (\text{<PAST SEES1>} \ \mathbf{s1} \ a \ \text{<THE } \mathbf{d1} \ (\text{DOG1 } \mathbf{d1})\text{>})) \ (\text{NAME } \mathbf{j1} \ \text{"Jill"}))$$

There is an inverse operation to lambda reduction, called lambda abstraction, that could be used to find a match. But the problem is that there are three possible lambda abstractions of the logical form, namely

$$(\hat{U} \ e \ (\text{<PAST SEES1>} \ e \ (\text{NAME } \mathbf{j1} \ \text{"Jill"}) \ \text{<THE } \mathbf{d1} \ (\text{DOG1 } \mathbf{d1})\text{>}))$$

$$(\hat{U} \ a \ (\text{<PAST SEES1>} \ \mathbf{s1} \ a \ \text{<THE } \mathbf{d1} \ (\text{DOG1 } d)\text{>}))$$

$$(\hat{U} \ o \ (\text{<PAST SEES 1>} \ \mathbf{s1} \ (\text{NAME } \mathbf{j1} \ \text{"Jill"}) \ o))$$

There is nothing in rule 1 that indicates which is the right abstraction, but only the second will yield an appropriate sentence.

On the other hand, the approach using features, as in Grammar 9.14, is reversible because it retains in the features the information necessary to determine how the logical form was constructed. It is easiest to understand how it does this by considering an example.

In many ways parsing and realization are very similar processes. Both can be viewed as building a syntactic tree. A parser starts with the words and tries to find a tree that accounts for them and hence determine the logical form of the sentence, while a realizer starts with a logical form and tries to find a tree to account for it and hence

determine the words to realize it. This analogy suggests that the standard parsing algorithms might be adaptable to realization. For instance, you could imagine using the standard top-down algorithm, where rules are selected from the grammar based on whether they match the intended SEM structure, and the words in the sentence are selected in a left-to-right fashion. The problem is that this approach would be extremely inefficient. For example, consider using Grammar 9.14 to realize an S with the SEM

```
(<PAST SEES1> s1 (NAME j1 "Jill") <THE d1 (DOG1 d1)>)
```

Rule 1 matches trivially since its SEM feature has a variable as its value, ?sernvp. There are no other S rules, so this one is instantiated. The standard top-down algorithm would now attempt to generate the two subconstituents, namely

```
(NP SEM ?semsubj)
```

```
(VP SUBJ ?semsubj
```

```
    SEM (<PAST SEES1> s1 (NAME j1 "Jill") <THE d1 (DOG1 d1)>))
```

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 288]

Initialization: Set L to a list containing the constituent that you wish to generate.

Do until L contains no nonlexical constituents:

1. If L contains a constituent C that is marked as a nonlexical head,
- 2 Then use a rule in the grammar to rewrite C. Any variables in C that are bound in the rewrite should be instantiated throughout the entire list.
3. Else choose a nonlexical constituent C, giving preference to one whose SEM feature is bound, if one exists. Use a rule in the grammar to rewrite C. Any variables in C that are bound in the rewrite should be instantiated throughout the entire list.

Figure 9.15 A head-driven realization algorithm

But the SEM of the NP is unconstrained. The realizer could only proceed by randomly generating noun phrases and then attempting to realize the VP with each one and backtracking until an appropriate one is found. Worse than that, with a more general grammar, the algorithm may fall into an infinite loop. Clearly, this is an unworkable strategy.

One method for avoiding these problems is to expand the constituents in a different order. For instance, choosing what term will be the subject depends on decisions made about the verb and the structure of the verb phrase, such as whether the active or passive voice is used, and so on. So it makes sense to expand the verb phrase first and then generate the appropriate subject. In fact, a good general strategy is to expand the head constituents first and then fill in the others. Figure 9.15 gives a simple algorithm to do this. The realization algorithm operates on a list of constituents much like the basic top-down parser described in Chapter 3. It continues to rewrite constituents in this list until the list consists only of lexical constituents, at which point the words can be generated.

This algorithm can easily be generalized so that it searches all possible realizations based on the grammar using a backtracking technique. The reason it works well is that, by expanding the head constituents first, the algorithm moves quickly down to the lexical level and chooses the words that have the most influence on the structure of the overall sentence. Once the lexical head is chosen, most of the structure of the rest of the constituent is determined.

Consider this algorithm operating with Grammar 9.14 and the initial input

```
(S SEM (<PAST SEES1> s1 (NAME j1 "Jill") <THE d1 (DOG1 d1)>))
```

The S constituent is rewritten based on rule 1 in the grammar to produce the following constituent list:

```
(NP SEM ?semsubj)
```

```
(VP SUBJ ?semsubj
```

```
SEM (<PAST SEES1> s1 (NAME j1 "Jill") <THE d1 (DOG] d1)>))
```

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 289]

The nonlexical head constituents are indicated in italics. Thus the VP is expanded next. Only rule 3 will match the SEM structure. As a result of the match, the following variables are bound:

```
?semv <- <PAST SEES1>  
  
?v <- s1  
  
?semsubj <- (NAME j1 "Jill")  
  
?semnp <- <THE d1 (DOG1 d1)>
```

Thus you obtain the following list of constituents after rewriting the VP and instantiating the variables throughout the list:

```
(NP SEM (NAME j1 "Jill"))  
  
(V[_np] SEM <PAST SEES1>)  
  
(NP SEM <THE d1 (DOG1 d1)>)
```

Since there is no nonlexical head, the algorithm now picks any nonlexical constituent with a bound SEM, say the first NP. Only rule 5 will match, yielding the lexical constituent (NAME SEM "Jill") and producing the constituent list

```
(NAME SEM "Jill")  
  
(V[_np] SEM <PAST SEES1>)  
  
(NP SEM <THE d1 (DOG1 d1)>)
```

The remaining NP is selected next. Rule 6 matches and the subconstituents (ART SEM THE) and (CNP SEM DOG1) are generated, producing the constituent list

```
(NAME SEM "Jill")  
  
(V[_np] SEM <PAST SEES1>)  
  
(ART SEM THE)  
  
(CNP SEM DOG1)
```

The CNP constituent is selected next and rewritten as a common noun with SEM DOG1, and the algorithm is complete. The constituent list is now a sequence of lexical categories:

```
(NAME SEM "Jill")  
  
(VLnp] SEM <PAST SEES1>)
```

(ART **SEM** THE)

(N **SEM** DOG1)

It is simple to produce the sentence Jill saw the dog from the lexical constituents. The grammar used in this example was very small, so there is only one possible realization of this sentence. With a larger grammar, a wider range of forms would be possible. For instance, if only the SEM feature is specified, the realization program would randomly pick between active and passive sentences when allowed by the verb. In other cases it might pick between different subcategorization structures. For instance, the same logical form might be realized as Jill gave the dog to Jack or Jill gave Jack the dog. Depending on the

[Allen 1995: Chapter 9 - Linking Syntax and Semantics / 290]

number of word senses used and whether different words have senses in common, the realizer may also randomly pick between various lexical realizations of a logical form. For instance, there could be a logical form that could be realized by the sentence Jill gave the money to the Humane Society or Jack donated the money to the Humane Society.

Each of these variations may have different effects in context, but these distinctions are not captured in the logical form. To force particular realizations, you would have to specify other features in addition to the SEM feature. For

instance, you might set the VOICE feature to active to guarantee an active voice sentence.

>> [back](#)

Summary

This chapter developed a rule-by-rule model of semantic interpretation by introducing the new features SEM and VAR to encode semantic information. The representation depends on the use of lambda expressions, and every constituent has a well-formed semantic form. With two minor extensions to introduce unique VAR values and to perform lambda reduction, any chart parsing algorithm will compute a logical form as it builds the syntactic structure. A variant of this approach that uses additional features instead of lambda expressions was discussed. This representation produces a reversible grammar, which can be used for semantic realization as well as semantic interpretation. A head-driven realization algorithm was used to generate sentences from a specified logical form. This expands the head constituents first to determine the overall structure of the sentence before filling in the details on the other constituents.

Related Work and Further Readings

Most work in semantics in the last two decades has been influenced by the work of Montague (1974), who proposed that the semantics for natural languages could be defined compositionally in the same way as the semantics for formal languages (1974). He introduced several key concepts, the most influential for our purposes being the adoption of the lambda calculus. A good reference for Montague style semantics is Chierchia and McConnell-Ginet (1990). Dowty et al. (1981) and Partee et al. (1993) give more detailed coverage.

The compositional semantic interpreter described in this chapter draws from a body of work, with key references being Rosenschein and Shieber (1982), Schubert and Pelletier (1982), and GPSG (Gazdar et al., 1985). Other important references on compositional interpretation include Hirst (1987), McCord (1986), Saint-Dizier (1985), and Hwang and Schubert (1993b). The use of feature unification in place of lambda reduction is discussed by Pereira and Shieber (1987) and Moore (1989). A good example of this technique is (Alshawi, 1992).

There is a substantial literature in natural language generation, an area that deals with two main issues: deciding on the content of what should be said, and then realizing that content as a sentence. The algorithm in Section 9.7 addresses

the realization part only, as it assumes that a logical form has already been identified. By the time the logical form is identified, many of the most difficult problems in generation - such as deciding on the content and on the main predicates and mode of reference - must have already been resolved. For a good survey of work in generation, see the article by McDonald in Shapiro (1992) under the heading Natural Language Generation. For a good collection of recent work, see the special issue of *Computational Intelligence* 7,4 (1991) and the papers in Dale et al. (1990). The head-driven algorithm described here is a simplified version of that described by Shieber et al. (1990).

>> [back](#)

Exercises for Chapter 9

1. (easy) Simplify the following formulas using lambda reduction:

$$((\hat{U} \ x \ (P \ x)) \ A)$$

$$((\hat{U} \ x \ (x \ A)) \ (\hat{U} \ y \ (Q \ y)))$$

$$((\hat{U} \ x \ ((\hat{U} \ y \ (P \ y)) \ x)) \ A)$$

2. (easy) Using the interpretation rules defined in this chapter and defining any others that you need, give a detailed trace of the interpretation of' the sentence The man gave the apple to Bill. In particular, give the analysis of each constituent and show its SEM feature.

3. (medium) This question involves specifying the semantic interpretation rules to analyze the different senses of the verb roll, as in

We rolled the log into the river.

The log rolled by the house.

The cook rolled the pastry with a large jar.

The ball rolled around the room.

We rolled the piano to the house on a dolly.

- a. Identify the different senses of the verb *roll* in the preceding sentences, and give an informal definition of each meaning. (You may use a dictionary if you wish.) Try to identify how each different sense allows different conclusions to be made from each sentence. Specify the lexical entries for the verb *roll*.
 - b. List the VP rules in this chapter that are needed to handle these sentences, and add any new rules as necessary.
 - c. Given the rules outlined in Exercise 3b, and assuming appropriate rules exist for the NPs, draw the parse tree with the logical forms shown for each constituent.
4. (medium) Draw the parse trees showing the semantic interpretations for the constituents for the following questions. Give the lexical entries showing

the SEM feature for each word used that is not defined in this chapter, and define any additional rules needed that are not specified in this chapter.

Who saw the dog?

Who did John give the book to?

5. (medium) Consider the following rules that are designed to handle verbs that take infinitive complements.

1. (VP **VAR** ?v **SEM** (\hat{U} ag (?semv ag (?semvp ag)))) \rightarrow

(V **SUBCAT** _vp-inf **SEM** ?semv)

(VP **VFORM** inf **SEM** ?semvp)

2. (VP **VAR** ?v **SEM** (\hat{U} ag (?semv ag (?semvp ?semnp)))) \rightarrow

(V **SUBCAT** _np_vp-inf **SEM** ?semv)

(NP **SEM** ?semnp)

(VP **VFORM** inf **SEM** ?semvp)

3. (VP **VFORM** inf ?v **SEM** ?semvp) \rightarrow

TO (VP **VFORM** base **SEM** ?semvp)

Note that the semantic structures built identify the subject of the embedded VP. For vp-inf verbs, rule 1, the implicit subject is the subject of the sentence, whereas for jip..vp-inf verbs, rule 2, the implicit subject is the object of the main verb. Using these rules and the rules described in this chapter, what logical forms are produced for the following sentences?

Jill wants to cry.

Jill wants Jack to cry.

Who does Jill want to see?

6. (medium) Consider the following sentences.

John is happy.

John is in the corner.

John is a man with a problem.

Each of these asserts some property of John. This suggests that the phrases happy, in the corner, and a man with a problem all should have similar semantic structures, presumably a unary predicate. In some approaches, the

PRED feature is extended to forms other than PPs as a way of capturing the similarities between all three of these cases. Assuming this approach, what should the interpretation of these three phrases be when they have the +PRED feature, as in this context? Write out the necessary rules to add to the grammar described in this chapter so that each sentence is handled appropriately.

>> [back](#)