In which we see why agents might want to exchange information-carrying messages with each other and how they can do so.

It is dusk in the savanna woodlands of Amboseli National Park near the base of Kilimanjaro. A group of vervet monkeys are foraging for food when one lets out a loud barking call. The others in the group recognize this as the leopard warning call (distinct from the short cough used to warn of eagles, or the chatter for snakes) and scramble for the trees. The vervet has successfully communicated with the group.

Communication is the intentional exchange of information brought about by the production and perception of signs drawn from a shared system of conventional signs. Most animals use signs to represent important messages: food here, predator nearby, approach, withdraw, let’s mate. In a partially observable world, communication can help agents be successful because they can learn information that is observed or inferred by others.

What sets humans apart from other animals is the complex system of structured messages known as language that enables us to communicate most of what we know about the world. Although chimpanzees, dolphins, and other mammals have shown vocabularies of hundreds of signs and some aptitude for stringing them together, only humans can reliably communicate an unbounded number of qualitatively different messages.

Of course, there are other attributes that are uniquely human: no other species wears clothes, creates representational art, or watches three hours of television a day. But when Turing proposed his test (see Section 1.1), he based it on language, because language is intimately tied to thinking. In this chapter, we will both explain how a communicating agent works and describe a fragment of English.

22.1 Communication as Action

One of the actions available to an agent is to produce language. This is called a speech act. “Speech” is used in the same sense as in “free speech,” not “talking,” so e-mailing, skywriting, and using sign language all count as speech acts. English has no neutral word for an agent that produces language by any means, so we will use speaker, hearer, and utterance as generic.
terms referring to any mode of communication. We will also use the term word to refer to any kind of conventional communicative sign.

Why would an agent bother to perform a speech act when it could be doing a “regular” action? We saw in Chapter 12 that agents in a multiagent environment can use communication to help arrive at joint plans. For example, a group of agents exploring the wumpus world together gains an advantage (collectively and individually) by being able to do the following:

- **Query** other agents about particular aspects of the world. This is typically done by asking questions: *Have you smelled the wumpus anywhere?*
- **Inform** each other about the world. This is done by making representative statements: *There’s a breeze here in 3 4.* Answering a question is another kind of informing.
- **Request** other agents to perform actions: *Please help me carry the gold.* Sometimes an **indirect speech act** (a request in the form of a statement or question) is considered more polite: *I could use some help carrying this.* An agent with authority can give commands (*Alpha go right; Bravo and Charlie go left*), and an agent with power can make a threat (*Give me the gold, or else*). Together, these kinds of speech acts are called **directives**.
- **Acknowledge** requests: *OK.*
- **Promise** or commit to a plan: *I’ll shoot the wumpus; you grab the gold.*

All speech acts affect the world by making air molecules vibrate (or the equivalent effect in some other medium) and thereby changing the mental state and eventually the future actions of other agents. Some kinds of speech acts transfer information to the hearer, assuming that the hearer’s decision making will be suitably affected by that information. Others are aimed more directly at making the hearer take some action. Another class of speech act, the **declarative**, appears to have a more direct effect on the world, as in *I now pronounce you man and wife or Strike three, you’re out.* Of course, the effect is achieved by creating or confirming a complex web of mental states among the agents involved: being married and being out are states characterized primarily by convention rather than by “physical” properties of the world.

The communicating agent’s task is to decide when a speech act of some kind is called for and which speech act, out of all the possibilities, is the right one. The problem of understanding speech acts is much like other **understanding** problems, such as understanding images or diagnosing illnesses. We are given a set of ambiguous inputs, and from them we have to work backwards to decide what state of the world could have created these inputs. However, because speech is a planned action, understanding it also involves plan recognition.

### Fundamentals of language

A **formal language** is defined as a (possibly infinite) set of **strings**. Each string is a concatenation of **terminal symbols**, sometimes called words. For example, in the language of first-order logic, the terminal symbols include \( \land \) and \( P \), and a typical string is \( P \land Q \). The string \( P \land Q \land \) is not a member of the language. Formal languages such as first-order logic and Java have strict mathematical definitions. This is in contrast to **natural languages**, such as Chinese, Danish, and English, that have no strict definition but are used by a community.
of speakers. For this chapter we will attempt to treat natural languages as if they were formal languages, although we recognize the match will not be perfect.

A grammar is a finite set of rules that specifies a language. Formal languages always have an official grammar, specified in manuals or books. Natural languages have no official grammar, but linguists strive to discover properties of the language by a process of scientific inquiry and then to codify their discoveries in a grammar. To date, no linguist has succeeded completely. Note that linguists are scientists, attempting to define a language as it is. There are also prescriptive grammarians who try to dictate how a language should be. They create rules such as “Don’t split infinitives” which are sometimes printed in style guides, but have little relevance to actual language usage.

Both formal and natural languages associate a meaning or semantics to each valid string. For example, in the language of arithmetic, we would have a rule saying that if “X” and “Y” are expressions, then “X + Y” is also an expression, and its semantics is the sum of X and Y. In natural languages, it is also important to understand the pragmatics of a string: the actual meaning of the string as it is spoken in a given situation. The meaning is not just in the words themselves, but in the interpretation of the words in situ.

Most grammar rule formalisms are based on the idea of phrase structure—that strings are composed of substrings called phrases, which come in different categories. For example, the phrases “the wumpus,” “the king,” and “the agent in the corner” are all examples of the category noun phrase, or NP. There are two reasons for identifying phrases in this way. First, phrases usually correspond to natural semantic elements from which the meaning of an utterance can be constructed; for example, noun phrases refer to objects in the world. Second, categorizing phrases helps us to describe the allowable strings of the language. We can say that any of the noun phrases can combine with a verb phrase (or VP) such as “is dead” to form a phrase of category sentence (or S). Without the intermediate notions of noun phrase and verb phrase, it would be difficult to explain why “the wumpus is dead” is a sentence whereas “wumpus the dead is” is not.

Category names such as NP, VP, and S are called nonterminal symbols. Grammars define nonterminals using rewrite rules. We will adopt the Backus–Naur form (BNF) notation for rewrite rules, which is described in Appendix B on page 984. In this notation, the meaning of a rule such as

\[ S \rightarrow NP \ VP \]

is that an S may consist of any NP followed by any VP.

The component steps of communication

A typical communication episode, in which speaker S wants to inform hearer H about proposition P using words W, is composed of seven processes:

Intention. Somehow, speaker S decides that there is some proposition P that is worth saying to hearer H. For our example, the speaker has the intention of having the hearer know that the wumpus is no longer alive.

Generation. The speaker plans how to turn the proposition P into an utterance that makes it likely that the hearer, upon perceiving the utterance in the current situation, can infer
Generative capacity

Grammatical formalisms can be classified by their generative capacity: the set of languages they can represent. Chomsky (1957) describes four classes of grammatical formalisms that differ only in the form of the rewrite rules. The classes can be arranged in a hierarchy, where each class can be used to describe all the languages that can be described by a less powerful class, as well as some additional languages. Here we list the hierarchy, most powerful class first:

Recursive enumerability grammars use unrestricted rules: both sides of the rewrite rules can have any number of terminal and nonterminal symbols, as in the rule \( A \ B \rightarrow C \). These grammars are equivalent to Turing machines in their expressive power.

Context-sensitive grammars are restricted only in that the right-hand side must contain at least as many symbols as the left-hand side. The name "context-sensitive" comes from the fact that a rule such as \( A \ S \ B \rightarrow A \ X \ B \) says that an \( S \) can be rewritten as an \( X \) in the context of a preceding \( A \) and a following \( B \). Context-sensitive grammars can represent languages such as \( a^n b^n c^n \) (a sequence of \( n \) copies of \( a \) followed by the same number of \( b \)s and then \( c \)s).

In context-free grammars (or CFGs), the left-hand side consists of a single nonterminal symbol. Thus, each rule licenses rewriting the nonterminal as the right-hand side in any context. CFGs are popular for natural language and programming language grammars, although it is now widely accepted that at least some natural languages have constructions that are not context-free (Pullum, 1991). Context-free grammars can represent \( a^n b^n \), but not \( a^n b^n c^n \).

Regular grammars are the most restricted class. Every rule has a single nonterminal on the left-hand side and a terminal symbol optionally followed by a nonterminal on the right-hand side. Regular grammars are equivalent in power to finite-state machines. They are poorly suited for programming languages, because they cannot represent constructs such as balanced opening and closing parentheses (a variation of the \( a^n b^n \) language). The closest they can come is representing \( a^* b^* \), a sequence of any number of \( a \)s followed by any number of \( b \)s.

The grammars higher up in the hierarchy have more expressive power, but the algorithms for dealing with them are less efficient. Up to the mid 1980s, linguists focused on context-free and context-sensitive languages. Since then, there has been increased emphasis on regular grammars, brought about by the need to process megabytes and gigabytes of online text very quickly, even at the cost of a less complete analysis. As Fernando Pereira put it, "The older I get, the further down the Chomsky hierarchy I go." To see what he means, compare Pereira and Warren (1980) with Mohri, Pereira, and Riley (2002).
the meaning $P$ (or something close to it). Assume that the speaker is able to come up with the words "The wumpus is dead," and call this $W$.

**Synthesis.** The speaker produces the physical realization $W'$ of the words $W$. This can be via ink on paper, vibrations in air, or some other medium. In Figure 22.1, we show the agent synthesizing a string of sounds $W'$ written in the phonetic alphabet defined on page 569: "[thaxwahmpaxshizdehd]." The words are run together; this is typical of quickly spoken speech.

**Perception.** $H$ perceives the physical realization $W'$ as $W'_2$ and decodes it as the words $W_2$. When the medium is speech, the perception step is called **speech recognition**; when it is printing, it is called **optical character recognition**. Both have moved from being esoteric to being commonplace in the 1990s, due largely to increased desktop computing power.

**Analysis.** $H$ infers that $W_2$ has possible meanings $P_1, \ldots, P_n$. We divide analysis into three main parts: syntactic interpretation (or parsing), semantic interpretation, and pragmatic interpretation. Parsing is the process of building a parse tree for an input string, as shown in Figure 22.1. The interior nodes of the parse tree represent phrases and the leaf nodes represent words. Semantic interpretation is the process of extracting the meaning of an utterance as an expression in some representation language. Figure 22.1 shows two possible semantic interpretations: that the wumpus is not alive and that it is tired (a colloquial meaning of dead). Utterances with several possible interpretations are said to be ambiguous. Pragmatic interpretation takes into account the fact that the same words can have different meanings in

---

**Figure 22.1** Seven processes involved in communication, using the example sentence "The wumpus is dead."
different situations. Whereas syntactic interpretation is a function of one argument, the string, pragmatic interpretation is a function of the utterance and the context or situation in which it is uttered. In the example, pragmatics does two things: replace the constant Now with the constant S3, which stands for the current situation, and replace Wumpus with Wumpus1, which stands for the single Wumpus that is known to be in this cave. In general, pragmatics can contribute much more to the final interpretation of an utterance; consider "I'm looking at the diamond" when spoken by a jeweler or by a baseball player. In Section 22.7, we will see that pragmatics allows us to interpret "It is dead" as meaning that the wumpus is dead if we are in a situation where the wumpus is salient.

Disambiguation. H infers that S intended to convey $P_i$ (where ideally $P_i = P$). Most speakers are not intentionally ambiguous, but most utterances have several feasible interpretations. Communication works because the hearer does the work of figuring out which interpretation is the one the speaker probably meant to convey. Notice that this is the first time we have used the word probably, and disambiguation is the first process that depends heavily on uncertain reasoning. Analysis generates possible interpretations; if more than one interpretation is found, then disambiguation chooses the one that is best.

Incorporation. $H$ decides to believe $P_i$ (or not). A totally naive agent might believe everything it hears, but a sophisticated agent treats the speech act as evidence for $P_i$, not confirmation of it.

Putting it all together, we get the agent program shown in Figure 22.2. Here the agent acts as a robot slave that can be commanded by a master. On each turn, the slave will answer a question or obey a command if the master has made one, and it will believe any statements made by the master. It will also comment (once) on the current situation if it has nothing more pressing to do, and it will plan its own action if left alone. Here is a typical dialog:

<table>
<thead>
<tr>
<th>ROBOT SLAVE</th>
<th>MASTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>I feel a breeze.</td>
<td>Go to 1 2.</td>
</tr>
<tr>
<td>Nothing is here.</td>
<td>Go north.</td>
</tr>
<tr>
<td>I feel a breeze and I smell a stench</td>
<td></td>
</tr>
<tr>
<td>and I see a glitter.</td>
<td>Grab the gold.</td>
</tr>
</tbody>
</table>

22.2 A Formal Grammar for a Fragment of English

In this section, we define a formal grammar for a small fragment of English that is suitable for making statements about the wumpus world. We will call this language $\mathcal{E}_0$. Later sections will improve on $\mathcal{E}_0$ to make it somewhat closer to real English. We are unlikely ever to devise a complete grammar for English, if only because no two persons would agree entirely on what constitutes valid English.

The Lexicon of $\mathcal{E}_0$

First we define the lexicon, or list of allowable words. The words are grouped into the categories or parts of speech familiar to dictionary users: nouns, pronouns, and names to denote
things, verbs to denote events, adjectives to modify nouns, and adverbs to modify verbs. Categories that are perhaps less familiar to some readers are articles (such as the), prepositions (in), and conjunctions (and). Figure 22.3 shows a small lexicon.

Each of the categories ends in . . . to indicate that there are other words in the category. However, it should be noted that there are two distinct reasons for the missing words. For nouns, verbs, adjectives, and adverbs, it is in principle infeasible to list them all. Not only are there tens of thousands of members in each class, but new ones—like MP3 or anime—are being added constantly. These four categories are called open classes. The other categories (pronoun, article, preposition, and conjunction) are called closed classes. They have a small number of words (a few to a few dozen) that can in principle be enumerated in full. Closed classes change over the course of centuries, not months. For example, "thee" and "thou" were commonly used pronouns in the 17th century, were on the decline in the 19th, and are seen today only in poetry and some regional dialects.

The Grammar of $E_0$

The next step is to combine the words into phrases. We will use five nonterminal symbols to define the different kinds of phrases: sentence ($S$), noun phrase ($NP$), verb phrase ($VP$),
Section 22.2. A Formal Grammar for a Fragment of English

Noun $\rightarrow$ stench | breeze | glitter | nothing | agent
   | wumpus | pit | pits | gold | east | ...
Verb $\rightarrow$ is | see | smell | shoot | feel | stinks
   | go | grab | carry | kill | turn | ...
Adjective $\rightarrow$ right | left | east | dead | back | smelly | ...
Adverb $\rightarrow$ here | there | near | by | ahead
   | right | left | east | south | back | ...
Pronoun $\rightarrow$ me | you | I | it | ...
Name $\rightarrow$ John | Mary | Boston | Aristotle | ...
Article $\rightarrow$ the | a | an | ...
Preposition $\rightarrow$ to | in | on | near | ...
Conjunction $\rightarrow$ and | or | but | ...
Digit $\rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Figure 22.3 The lexicon for $E_0$.

$S \rightarrow NP \ VP$  I feel a breeze
   | $S$ Conjunction $S$  I feel a breeze + and + I smell a wumpus

NP $\rightarrow$ Pronoun  I
   | Name  John
   | Noun  pits
   | Article Noun  the + wumpus
   | Digit Digit  3 4
   | NP PP  the wumpus + to the east
   | NP RelClause  the wumpus + that is smelly

VP $\rightarrow$ Verb  stinks
   | VP NP  feel + a breeze
   | VP Adjective  is + smelly
   | VP PP  turn + to the east
   | VP Adverb  go + ahead

PP $\rightarrow$ Preposition NP  to + the east
RelClause $\rightarrow$ that VP  that + is smelly

Figure 22.4 The grammar for $E_0$, with example phrases for each rule.
prepositional phrase (PP), and relative clause (RelClause). Figure 22.4 shows a grammar for \( E_0 \), with an example for each rewrite rule. \( E_0 \) generates good English sentences such as the following:

- John is in the pit
- The wumpus that stinks is in 2 2
- Mary is in Boston and John stinks

Unfortunately, the grammar overgenerates: that is, it generates sentences that are not grammatical, such as “Me go Boston” and “I smell pit gold wumpus nothing east.” It also undergenerates: there are many sentences of English that it rejects, such as “I think the wumpus is smelly.” (Another shortcoming is that the grammar does not capitalize the first word of a sentence, nor add punctuation at the end. That is because it is designed primarily for speech, not writing.)

### 22.3 Syntactic Analysis (Parsing)

We have already defined parsing as the process of finding a parse tree for a given input string. That is, a call to the parsing function \( \text{PARSE} \), such as

\[
\text{PARSE}("\text{the wumpus is dead"}, E_0, S)
\]

should return a parse tree with root \( S \) whose leaves are “the wumpus is dead” and whose internal nodes are nonterminal symbols from the grammar \( E_0 \). You can see such a tree in Figure 22.1. In linear text, we write the tree as

\[
[S: [NP: [Article: the][Noun: wumpus]] [VP: [Verb: is][Adjective: dead]]]
\]

Parsing can be seen as a process of searching for a parse tree. There are two extreme ways of specifying the search space (and many variants in between). First, we can start with the \( S \) symbol and search for a tree that has the words as its leaves. This is called top-down parsing (because the \( S \) is drawn at the top of the tree). Second, we could start with the words and search for a tree with root \( S \). This is called bottom-up parsing. Top-down parsing can be precisely defined as a search problem as follows:

- **The initial state** is a parse tree consisting of the root \( S \) and unknown children: \([S: ?]\). In general, each state in the search space is a parse tree.
- **The successor function** selects the leftmost node in the tree with unknown children. It then looks in the grammar for rules that have the root label of the node on the left-hand side. For each such rule, it creates a successor state where the ? is replaced by a list corresponding to the right-hand side of the rule. For example, in \( E_0 \) there are two rules for \( S \), so the tree \([S: ?]\) would be replaced by the following two successors:

---

1. A relative clause follows and modifies a noun phrase. It consists of a relative pronoun (such as “who” or “that”) followed by a verb phrase. (Another kind of relative clause is discussed in exercise 22.12.) An example of a relative clause is \( \text{that stinks} \) in “The wumpus \( \text{that stinks} \) is in 2 2.”
2. The reader might notice that top-down and bottom-up parsing are analogous to backward and forward chaining, respectively, as described in Chapter 7. We will see shortly that the analogy is exact.
Section 22.3. Syntactic Analysis (Parsing)

\[
[S: ?] [\textbf{Conjunction}: ?] [S: ?] \\
[S: [NP: ?] [VP: ?]]
\]

The second of these has seven successors, one for each rewrite rule of \(NP\).

- The **goal test** checks that the leaves of the parse tree correspond exactly to the input string, with no unknowns and no uncovered inputs.

One big problem for top-down parsing is dealing with so-called **left-recursive rules**—that is, rules of the form \(X \rightarrow X \ldots\). With a depth-first search, such a rule would lead us to keep replacing \(X\) with \([X: X \ldots]\) in an infinite loop. With a breadth-first search we would successfully find parses for valid sentences, but when given an invalid sentence, we would get stuck in an infinite search space.

The formulation of bottom-up parsing as a search is as follows:

- The **initial state** is a list of the words in the input string, each viewed as a parse tree that is just a single leaf node—for example; \([\text{the, wumpus, is, dead}]\). In general, each state in the search space is a list of parse trees.

- The **successor function** looks at every position \(i\) in the list of trees and at every right-hand side of a rule in the grammar. If the subsequence of the list of trees starting at \(i\) matches the right-hand side, then the subsequence is replaced by a new tree whose category is the left-hand side of the rule and whose children are the subsequence. By “matches,” we mean that the category of the node is the same as the element in the right-hand side. For example, the rule \(\text{Article} \rightarrow \text{the}\) matches the subsequence consisting of the first node in the list \([\text{the, wumpus, is, dead}]\), so a successor state would be \([\text{Article: the}, \text{wumpus, is, dead}]\).

- The **goal test** checks for a state consisting of a single tree with root \(S\).

See Figure 22.5 for an example of bottom-up parsing.

<table>
<thead>
<tr>
<th>step</th>
<th>list of nodes</th>
<th>subsequence</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>the wumpus is dead</td>
<td>the</td>
<td>Article (\rightarrow) the</td>
</tr>
<tr>
<td>2</td>
<td>Article wumpus is dead</td>
<td>wumpus</td>
<td>Noun (\rightarrow) wumpus</td>
</tr>
<tr>
<td>3</td>
<td>Article Noun is dead</td>
<td>Article Noun</td>
<td>(NP \rightarrow) Article Noun</td>
</tr>
<tr>
<td>4</td>
<td>NP is dead</td>
<td>is</td>
<td>Verb (\rightarrow) is</td>
</tr>
<tr>
<td>5</td>
<td>NP Verb dead</td>
<td>dead</td>
<td>Adjective (\rightarrow) dead</td>
</tr>
<tr>
<td>6</td>
<td>NP Verb Adjective</td>
<td>Verb</td>
<td>VP (\rightarrow) Verb</td>
</tr>
<tr>
<td>7</td>
<td>NP VP Adjective</td>
<td>VP Adjective</td>
<td>(VP \rightarrow) VP Adjective</td>
</tr>
<tr>
<td>8</td>
<td>NP VP</td>
<td>NP VP</td>
<td>(S \rightarrow) NP VP</td>
</tr>
</tbody>
</table>

FIGURE 22.5 Trace of a bottom up parse on the string “The wumpus is dead.” We start with a list of nodes consisting of words. Then we replace subsequences that match the right-hand side of a rule with a new node whose root is the left-hand side. For example, in the third line the \(\text{Article}\) and \(\text{Noun}\) nodes are replaced by an \(NP\) node that has those two nodes as children. The top-down parse would produce a similar trace, but in the opposite direction.
Both top-down and bottom-up parsing can be inefficient, because of the multiplicity of ways in which multiple parses for different phrases can be combined. Both can waste time searching irrelevant portions of the search space. Top-down parsing can generate intermediate nodes that could never be latched by the words, and bottom-up parsing can generate partial parses of the words that could not appear in an $S$.

Even if we had a perfect heuristic function that allowed us to search without any irrelevant digressions, these algorithms would still be inefficient, because some sentences have exponentially many parse trees. The next subsection shows what to do about that.

**Efficient parsing**

Consider the following two sentences:

Have the students in section 2 of Computer Science 101 take the exam.

Have the students in section 2 of Computer Science 101 taken the exam?

Even though they share the first 10 words, these sentences have very different parses, because the first is a command and the second is a question. A left-to-right parsing algorithm would have to guess whether the first word is part of a command or a question and will not be able to tell if the guess is correct until at least the eleventh word, *take* or *taken*. If the algorithm guesses wrong, it will have to backtrack all the way to the first word. This kind of backtracking is inevitable, but if our parsing algorithm is to be efficient, it must avoid reanalyzing "the students in section 2 of Computer Science 101" as an $NP$ each time it backtracks.

In this section, we will develop a parsing algorithm that avoids this source of inefficiency. The basic idea is an example of **dynamic programming**: *every time we analyze a substring, store the results so we won't have to re-analyze it later*. For example, once we discover that "the students in section 2 of Computer Science 101" is an $NP$, we can record that result in a data structure known as a **chart**. Algorithms that do this are called **chart parsers**. Because we are dealing with context-free grammars, any phrase that was found in the context of one branch of the search space can work just as well in any other branch of the search space.

The chart for an $n$-word sentence consists of $n + 1$ **vertices** and a number of **edges** that connect vertices. Figure 22.6 shows a chart with six vertices (circles) and three edges (lines). For example, the edge labeled

$$[0, 5, S \rightarrow NP \ VP \ .]$$

means that an $NP$ followed by a $VP$ combine to make an $S$ that spans the string from 0 to 5. The symbol $.$ in an edge separates what has been found so far from what remains to be found.\(^3\) Edges with $.$ at the end are called **complete edges**. The edge

$$[0, 2, S \rightarrow NP \ . \ VP]$$

says that an $NP$ spans the string from 0 to 2 (the first two words) and that if we could find a $VP$ to follow it, then we would have an $S$. Edges like this with the dot before the end are called **incomplete edges**, and we say that the edge is looking for a $VP$.

\(^3\) It is because of the $.$ that edges are sometimes called **dotted rules**.
function CHART-PARSE(words, grammar) returns chart

chart ← array[0... LENGTH(words)] of empty lists
ADD-EDGE([0, 0, S' → • S])
for i ← from 0 to LENGTH(words) do
  SCANNER(i, words[i])
return chart

procedure ADD-EDGE(edge)
  /* Add edge to chart, and see if it extends or predicts another edge. */
  if edge not in chart[END(edge)] then
    append edge to chart[END(edge)]
    if edge has nothing after the dot then EXTENDER(edge)
    else PREDICTOR(edge)
  
procedure SCANNER(j, word)
  /* For each edge expecting a word of this category here, extend the edge. */
  for each [i, j, A → α • B β] in chart[j] do
    if word is of category B then
      ADD-EDGE([i, j+1, A → α B • β])
  
procedure PREDICTOR([i, j, A → α • B β])
  /* Add to chart any rules for B that could help extend this edge */
  for each (B → γ) in REWRITES-FOR(B, grammar) do
    ADD-EDGE([i, j, B → • γ])

procedure EXTENDER([j, k, B → γ •])
  /* See what edges can be extended by this edge */
  eB ← the edge that is the input to this procedure
  for each [i, j, A → α • B' β] in chart[j] do
    if B = B' then
      ADD-EDGE([i, k, A → α eB • β])

Figure 22.6  Part of the chart for the sentence "The agent feels a breeze." All six vertices are shown, but only three of the edges that would make up a complete parse.

Figure 22.7  The chart-parsing algorithm. S is the start symbol and S' is a new nonterminal symbol. chart[j] is the list of edges that end at vertex j. The Greek letters match a string of zero or more symbols.
Figure 22.7 shows the chart-parsing algorithm. The main idea is to combine the best of top-down and bottom-up parsing. The procedure Predictor is top-down: it makes entries into the chart that say what symbols are desired at what locations. Scanner is the bottom-up procedure that starts from the words, but it will use a word only to extend an existing chart entry. Similarly, Extender builds constituents bottom-up, but only to extend an existing chart entry.

We use a trick to start the whole algorithm: we add the edge \([0, 0, \text{\$S'} \rightarrow \text{\$S}]\) to the chart, where \(\text{\$S}\) is the grammar's start symbol, and \(\text{\$S'}\) is a new symbol that we just invented. The call to Add-Edge causes the Predictor to add edges for the rules that can yield an \(\text{\$S}\)—that is, \([\text{\$S} \rightarrow \text{NP} \text{VP}]\). Then we look at the first constituent of that rule, \(\text{NP}\), and add rules for every way to yield an \(\text{NP}\). Eventually, the predictor adds, in a top-down fashion, all possible edges that could be used in the service of creating the final \(\text{\$S}\).

When the predictor for \(\text{\$S'}\) is finished, we enter a loop that calls Scanner for each word in the sentence. If the word at position \(j\) is a member of a category \(B\) that some edge is looking for at \(j\), then we extend that edge, noting the word as an instance of \(B\). Notice that each call to Scanner can end up calling Predictor and Extender recursively, thereby interleaving the top-down and bottom-up processing.

The other bottom-up component, Extender,\(^4\) takes a complete edge with left hand side \(B\) and uses it to extend any incomplete rule in the chart that ends where the complete edge starts if the incomplete rule is looking for a \(B\).

Figures 22.8 and 22.9 show a chart and trace of the algorithm parsing the sentence "I feel it" (which is an answer to the question "Do you feel a breeze?"). Thirteen edges (labeled a–m) are recorded in the chart, including five complete edges (shown above the vertices of the chart) and eight incomplete ones (below the vertices). Note the cycle of predictor, scanner, and extender actions. For example, the predictor uses the fact that edge \(a\) is looking for an \(\text{\$S}\) to license the prediction of an \(\text{NP}\) (edge \(b\)) and then a Pronoun (edge \(c\)). Then the scanner recognizes that there is a Pronoun in the right place (edge \(d\)), and the extender combines the incomplete edge \(b\) with the complete edge \(d\) to yield a new edge, \(e\).

The chart-parsing algorithm avoids building a large class of edges that would have been examined by the simple bottom-up procedure. Consider the sentence "The ride the horse gave was wild." A bottom-up parse would label "ride the horse" as a VP and then discard the parse tree when it is found not to fit into a larger \(\text{\$S}\). But \(E_0\) does not allow a VP to follow "the," so the chart-parsing algorithm will never predict a VP at that point and thus will avoid wasting time building the VP constituent there. Algorithms that work from left to right and avoid building these impossible constituents are called left-corner parsers, because they build up a parse tree that starts with the grammar's start symbol and extends down to the leftmost word in the sentence (the left corner). An edge is added to the chart only if it can serve to extend this parse tree. (See Figure 22.10 for an example.)

The chart parser uses only polynomial time and space. It requires \(O(kn^2)\) space to store the edges, where \(n\) is the number of words in the sentence and \(k\) is a constant that depends

\(^4\) Traditionally, our Extender Procedure has been called Completer. This name is misleading, because the procedure does not complete edges: it takes a complete edge as input and extends incomplete edges.
Section 22.3.  Syntactic Analysis (Parsing)

Figure 22.8  Chart for a parse of “0 I 1 feel 2 it 3.” The notation \( m:S \) means that edge \( m \) has an \( S \) on the left-hand side, while the notation \( f:VP/Verb \) means that edge \( f \) has a \( VP \) on the left-hand side, but is looking for a \( Verb \). There are five complete edges above the vertices and eight incomplete edges below.

<table>
<thead>
<tr>
<th>Edge</th>
<th>Procedure</th>
<th>Derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>INITIALIZER</td>
<td>([0, 0, S' \rightarrow \bullet S] )</td>
</tr>
<tr>
<td>b</td>
<td>PREDICTOR(a)</td>
<td>([0, 0, S \rightarrow \bullet NP VP] )</td>
</tr>
<tr>
<td>c</td>
<td>PREDICTOR(b)</td>
<td>([0, 0, NP \rightarrow \bullet Pronoun] )</td>
</tr>
<tr>
<td>d</td>
<td>SCANNER(c)</td>
<td>([0, 1, NP \rightarrow NP \bullet VP] )</td>
</tr>
<tr>
<td>e</td>
<td>EXTENDER(b,d)</td>
<td>([0, 1, S \rightarrow NP \bullet VP] )</td>
</tr>
<tr>
<td>f</td>
<td>PREDICTOR(e)</td>
<td>([1, 1, VP \rightarrow \bullet Verb] )</td>
</tr>
<tr>
<td>g</td>
<td>PREDICTOR(f)</td>
<td>([1, 1, VP \rightarrow VP \bullet NP] )</td>
</tr>
<tr>
<td>h</td>
<td>SCANNER(f)</td>
<td>([1, 2, VP \rightarrow Verb \bullet] )</td>
</tr>
<tr>
<td>i</td>
<td>EXTENDER(g,h)</td>
<td>([1, 2, VP \rightarrow VP \bullet NP] )</td>
</tr>
<tr>
<td>j</td>
<td>PREDICTOR(g)</td>
<td>([2, 2, NP \rightarrow \bullet Pronoun] )</td>
</tr>
<tr>
<td>k</td>
<td>SCANNER(j)</td>
<td>([2, 3, NP \rightarrow Pronoun \bullet] )</td>
</tr>
<tr>
<td>l</td>
<td>EXTENDER(i,k)</td>
<td>([1, 3, VP \rightarrow VP NP \bullet] )</td>
</tr>
<tr>
<td>m</td>
<td>EXTENDER(e,l)</td>
<td>([0, 3, S \rightarrow NP VP \bullet] )</td>
</tr>
</tbody>
</table>

Figure 22.9  Trace of a parse of “0 I 1 feel 2 it 3.” For each edge a-m, we show the procedure used to derive the edge from other edges already in the chart. Some edges were omitted for brevity.
Figure 22.10 A left-corner parsing algorithm avoids predicting a VP starting with “ride,” but does predict a VP starting with “was,” because the grammar expects a VP following an NP. The triangle over “the horse gave” means that the words have a parse as a RelClause, but with additional intermediate constituents that are not shown.

on the grammar. When it can build no more edges it stops, so we know that the algorithm terminates (even when there are left-recursive rules). In fact, it takes time $O(n^3)$ in the worst case, which is the best that can be achieved for context-free grammars. The bottleneck for CHART-Parse is EXTENDER, which must try to extend each of $O(n)$ incomplete edges ending at position $j$ with each of $O(n)$ complete edges starting at $j$, for each of $n + 1$ different values of $j$. Multiplying these together, we get $O(n^3)$. This gives us something of a paradox: how can an $O(n^3)$ algorithm return an answer that might contain an exponential number of parse trees? Consider an example: the sentence

“Fall leaves fall and spring leaves spring”

is ambiguous because each word (except “and”) can be either a noun or a verb, and “fall” and “spring” can be adjectives as well. Altogether, the sentence has four parses:

[S: [S: [NP: Fall leaves] fall] and [S: [NP: spring leaves] spring]];
[S: [S: [NP: Fall leaves] fall] and [S: [spring [VP: leaves spring]]];
[S: [S: Fall [VP: leaves fall]] and [S: [NP: spring leaves] spring]];
[S: [S: Fall [VP: leaves fall]] and [S: [spring [VP: leaves spring]]].

If we had $n$ ambiguous conjoined subsentences, we would have $2^n$ ways of choosing parses for the subsentences. How does the chart parser avoid exponential processing time? There are actually two answers. First, the CHART-Parse algorithm itself is actually a recognizer, not a parser. If there is a complete edge of the form $[0, n, S → α •]$ in the chart, then we have recognized $S$. Recovering the parse tree from this edge is not considered part of

---

5 The parse $[S: Fall [VP: leaves fall]]$ is equivalent to “Autumn abandons autumn.”

6 There also would be $O(n!)$ ambiguity in the way the components conjoin with each other—for example, $(X$ and $(Y$ and $Z))$ versus $((X$ and $Y)$ and $Z)$. But that is another story, one that is told quite well by Church and Patil (1982).
CHART-PARSE’s job, but it can be done. Note, in the last line of EXTENDER, that we build up α as a list of edges, εB, not just a list of category names. So to convert an edge into a parse tree, simply look recursively at the component edges, converting each \([i, j, X \rightarrow \alpha \bullet]\) into the tree \([X : \alpha]\). This is straightforward, but it gives us only one parse tree.

The second answer is that if you want all possible parses, you’ll have to dig deeper into the chart. While we’re converting the edge \([i, j, X \rightarrow \alpha \bullet]\) into the tree \([X : \alpha]\), we’ll also look to see whether there are any other edges of the form \([i, j, X \rightarrow \beta \bullet]\). If there are, these edges will generate additional parses. Now we have a choice of what to do with them. We could enumerate all the possibilities, and that means that the paradox would be resolved and we would require an exponential amount of time to list the parses. Or we could prolong the mystery a little longer and represent the parses with a structure called a packed forest, which looks like this:

\[
[S:\{[NP: \text{Fall leaves}] [VP: \text{fall}]\}] \quad \text{and} \quad [S:\{[NP: \text{spring leaves}] [VP: \text{spring}]\}]
\]

The idea is that each node can be either a regular parse tree node or a set of tree nodes. This enables us to return a representation of an exponential number of parses in a polynomial amount of space and time. Of course, when \(n = 2\), there is not much difference between \(2^n\) and \(2n\), but for large \(n\), such a representation offers considerable saving. Unfortunately, this simple packed forest approach won’t handle all the \(O(n!)\) ambiguity in how the conjunctions associate. Maxwell and Kaplan (1995) show how a more complex representation based on the principles of truth maintenance systems can pack the trees even tighter.

\[
\begin{align*}
S & \rightarrow NP_S \ VP \ | \ \ldots \\
NP_S & \rightarrow \text{Pronoun}_S \ | \ \text{Name} \ | \ \text{Noun} \ | \ \ldots \\
NP_O & \rightarrow \text{Pronoun}_O \ | \ \text{Name} \ | \ \text{Noun} \ | \ \ldots \\
VP & \rightarrow VP \ NP_O \ | \ \ldots \\
PP & \rightarrow \text{Preposition} \ NP_O \\
\text{Pronoun}_S & \rightarrow \text{I} \ | \ \text{you} \ | \ \text{he} \ | \ \text{she} \ | \ \text{it} \ | \ \ldots \\
\text{Pronoun}_O & \rightarrow \text{me} \ | \ \text{you} \ | \ \text{him} \ | \ \text{her} \ | \ \text{it} \ | \ \ldots \\
\end{align*}
\]

\[
\begin{align*}
S & \rightarrow NP(\text{Subjective}) \ VP \ | \ \ldots \\
NP(\text{case}) & \rightarrow \text{Pronoun}(\text{case}) \ | \ \text{Name} \ | \ \text{Noun} \ | \ \ldots \\
VP & \rightarrow VP \ NP(\text{Objective}) \ | \ \ldots \\
PP & \rightarrow \text{Preposition} \ NP(\text{Objective}) \\
\text{Pronoun}(\text{Subjective}) & \rightarrow \text{I} \ | \ \text{you} \ | \ \text{he} \ | \ \text{she} \ | \ \text{it} \ | \ \ldots \\
\text{Pronoun}(\text{Objective}) & \rightarrow \text{me} \ | \ \text{you} \ | \ \text{him} \ | \ \text{her} \ | \ \text{it} \ | \ \ldots \\
\end{align*}
\]

Figure 22.11 Top: A BNF grammar for the language \(E_1\), which handles subjective and objective cases in noun phrases and thus does not over-generate quite so badly. The portions that are identical to \(E_0\) have been omitted. Bottom: A definite clause grammar (DCG) of \(E_1\).
22.4 **AUGMENTED GRAMMARS**

We saw in Section 22.2 that the simple grammar for $\mathcal{E}_0$ generates "I smell a stench" and many other sentences of English. Unfortunately, it also generates many non-sentences such as "Me smell a stench." To avoid this problem, our grammar would have to know that "me" is not a valid $NP$ when it is the subject of a sentence. Linguists say that the pronoun "I" is in the subjective case, and "me" is in the objective case.\(^7\) When we take case into account, we realize that the $\mathcal{E}_0$ grammar is not context-free: it is not true that any $NP$ is equal to any other regardless of context. We can fix the problem by introducing new categories such as $NP_S$ and $NP_O$, to stand for noun phrases in the subjective and objective case, respectively.

We would also need to split the category *Pronoun* into the two categories $Pronoun_S$ (which includes "I") and $Pronoun_O$ (which includes "me"). The top part of Figure 22.11 shows the complete BNF grammar for case agreement; we call the resulting language $\mathcal{E}_1$. Notice that all the $NP$ rules must be duplicated, once for $NP_S$ and once for $NP_O$.

Unfortunately, $\mathcal{E}_1$ still overgenerates. English and many other languages require agreement between the subject and main verb of a sentence. For example, if "I" is the subject, then "I smell" is grammatical, but "I smells" is not. If "it" is the subject, we get the reverse. In English, the agreement distinctions are minimal: most verbs have one form for third-person singular subjects (he, she, or it), and a second form for all other combinations of person and number. There is one exception: "I am / you are / he is" has three forms. If we multiply these three distinctions by the two distinctions of $NP_S$ and $NP_O$, we end up with six forms of $NP$.

As we discover more distinctions, we end up with an exponential number.

The alternative is to augment the existing rules of the grammar instead of introducing new rules. We will first give an example of what we would like an augmented rule to look like (see the bottom half of Figure 22.11) and then formally define how to interpret the rules. Augmented rules allow for parameters on nonterminal categories. Figure 22.11 shows how to describe $\mathcal{E}_1$ using augmented rules. The categories $NP$ and *Pronoun* have a parameter indicating their case. (Nouns do not have case in English, although they do in many other languages.) In the rule for $S$, the $NP$ must be in the subjective case, whereas in the rules for $VP$ and $PP$, the $NP$ must be in the objective case. The rule for $NP$ takes a variable, case, as its argument. The intent is that the $NP$ can have any case, but if the $NP$ is rewritten as a *Pronoun*, then it must have the same case. This use of a variable—avoiding a decision where the distinction is not important—is what keeps the size of the rule set from growing exponentially with the number of features.

This formalism for augmentations is called **definite clause grammar** or DCG, because each grammar rule can be interpreted as a definite clause in Horn logic.\(^8\) First we will show how a normal, unaugmented rule can be interpreted as a definite clause. We consider each

---

\(^7\) The subjective case is also sometimes called the nominative case and the objective case is sometimes called the accusative case. Many languages also have a dative case for words in the indirect object position.

\(^8\) Recall that a definite clause, when written as an implication, has exactly one atom in its consequent, and a conjunction of zero or more atoms in its antecedent. Two examples are $A \land B \Rightarrow C$ and just $C$. 
category symbol to be a predicate on strings, so that $NP(s)$ is true if the string $s$ forms an $NP$. The CFG rule

$$S \rightarrow NP \ VP$$

is shorthand for the definite clause

$$NP(s_1) \land VP(s_2) \Rightarrow S(s_1 + s_2).$$

Here $s_1 + s_2$ denotes the concatenation of two strings, so this rule says that if the string $s_1$ is an $NP$ and the string $s_2$ is a $VP$, then their concatenation is an $S$, which is exactly how we were already interpreting the CFG rule. It is important to note that DCGs allow us to talk about parsing as logical inference. This makes it possible to reason about languages and strings in many different ways. For example, we can do bottom-up parsing using forward chaining or top-down parsing using backward chaining. We will see that it also means that we can use the same grammar for both parsing and generation.

The real benefit of the DCG approach is that we can augment the category symbols with additional arguments other than the string argument. For example, the rule

$$NP(case) \rightarrow Pronoun(case)$$

is shorthand for the definite clause

$$Pronoun(case, s_1) \Rightarrow NP(case, s_1).$$

This says that if the string $s_1$ is a $Pronoun$ with case specified by the variable $case$, then $s_1$ is also an $NP$ with the same case. In general, we can augment a category symbol with any number of arguments, and the arguments are parameters that are subject to unification as in regular Horn clause inference.

There is a price to pay for this convenience: we are providing the grammar writer with the full power of a theorem-prover, so we give up the guarantees of $O(n^3)$ syntactic parsing; parsing with augmentations can be NP-complete or even undecidable, depending on the augmentations.

A few more tricks are necessary to make DCG work; for example, we need a way to specify terminal symbols, and it is convenient to have a way not to add the automatic string argument. Putting everything together, we define definite clause grammar as follows:

- The notation $X \rightarrow Y \ Z \ldots$ translates as $Y(s_1) \land Z(s_2) \land \ldots \Rightarrow X(s_1 + s_2 + \ldots)$.
- The notation $X \rightarrow Y | Z | \ldots$ translates as $Y(s) \lor Z(s) \lor \ldots \Rightarrow X(s)$.
- In either of the preceding rules, any nonterminal symbol $Y$ can be augmented with one or more arguments. Each argument can be a variable, a constant, or a function of arguments. In the translation, these arguments precede the string argument (e.g., $NP(case)$ translates as $NP(case, s_1)$).
- The notation $\{P(\ldots)\}$ can appear on the right-hand side of a rule and translates verbatim into $P(\ldots)$. This allows the grammar writer to insert a test for $P(\ldots)$ without having the automatic string argument added.
- The notation $X \rightarrow \text{word}$ translates as $X([\text{word}])$.

The problem of subject-verb agreement could also be handled with augmentations, but we defer that to Exercise 22.2. Instead, we address a harder problem: verb subcategorization.
Verb subcategorization

$\mathcal{E}_1$ is an improvement over $\mathcal{E}_0$, but the $\mathcal{E}_1$ grammar still overgenerates. One problem is in the way verb phrases are put together. We want to accept verb phrases like “give me the gold” and “go to 1 2.” All these are in $\mathcal{E}_1$, but unfortunately so are “go me the gold” and “give to 1 2.” The language $\mathcal{E}_2$ eliminates these VPs by stating explicitly which phrases can follow which verbs. We call this list the subcategorization list for the verb. The idea is that the category Verb is broken into subcategories—one for verbs that have no object, one for verbs that take a single object, and so on.

To implement this idea, we give each verb a subcategorization list that lists the verb's complements. A complement is an obligatory phrase that follows the verb within the verb phrase. So in “Give the gold to me,” the NP “the gold” and the PP “to me” are complements of “give.”\footnote{This is one definition of complement, but other authors have different terminology. Some say that the subject of the verb is also a complement. Others say that only the prepositional phrase is a complement and that the noun phrase should be called an argument.} We would write this as

$$\text{Verb}([NP, PP]) \rightarrow \text{give} \mid \text{hand} \mid \ldots$$

It is possible for a verb to have several different subcategorizations, just as it is possible for a word to belong to several different categories. In fact, “give” also has the subcategorization list $[NP, NP]$, as in “Give me the gold.” We can treat this like any other kind of ambiguity. Figure 22.12 gives some examples of verbs and their subcategorization lists (or subcats for short).

![Figure 22.12 Examples of verbs with their subcategorization lists.](image)

<table>
<thead>
<tr>
<th>Verb</th>
<th>Subcats</th>
<th>Example Verb Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>give</td>
<td>$[NP, PP]$</td>
<td>give the gold in 3 3 to me</td>
</tr>
<tr>
<td></td>
<td>$[NP, NP]$</td>
<td>give me the gold</td>
</tr>
<tr>
<td>smell</td>
<td>$[NP]$</td>
<td>smell a wumpus</td>
</tr>
<tr>
<td></td>
<td>$[Adjective]$</td>
<td>smell awful</td>
</tr>
<tr>
<td></td>
<td>$[PP]$</td>
<td>smell like a wumpus</td>
</tr>
<tr>
<td>is</td>
<td>$[Adjective]$</td>
<td>is smelly</td>
</tr>
<tr>
<td></td>
<td>$[PP]$</td>
<td>is in 2 2</td>
</tr>
<tr>
<td></td>
<td>$[NP]$</td>
<td>is a pit</td>
</tr>
<tr>
<td>died</td>
<td>$[]$</td>
<td>died</td>
</tr>
<tr>
<td>believe</td>
<td>$[S]$</td>
<td>believe the wumpus is dead</td>
</tr>
</tbody>
</table>

To integrate verb subcategorization into the grammar, we take three steps. The first step is to augment the category VP to take a subcategorization argument, $VP(subcat)$, that indicates the list of complements that are needed to form a complete VP. For example, “give” can be made into a complete VP by adding $[NP, PP]$. “give the gold” can be made complete by adding $[PP]$, and “give the gold to me” is already a complete VP; therefore its
subcategorization list is the empty list, $[]$. That gives us these rules for $VP$:

$$VP(subcat) \rightarrow \begin{array}{l}
\text{Verb}(subcat) \\
| \hspace{1cm} VP(subcat + [NP]) \hspace{0.5cm} NP(\text{Objective}) \\
| \hspace{1cm} VP(subcat + [Adjective]) \hspace{0.5cm} Adjective \\
| \hspace{1cm} VP(subcat + [PP]) \hspace{0.5cm} PP.
\end{array}$$

The last line can be read as “A $VP$ with a given subcat list, $subcat$, can be formed by an
embedded $VP$ followed by a $PP$, as long as the embedded $VP$ has a subcat list that starts
with the elements of the list $subcat$ and ends with the symbol $PP$.” For example, a $VP([[]])$ is
formed by a $VP([PP])$ followed by a $PP$. The first line says that a $VP$ with subcategorization
list $subcat$ can be formed by a $Verb$ with the same subcategorization list. For example, a
$VP([NP])$ can be formed by a $Verb([NP])$. One example of such a verb is “grab,” so “grab
the gold” is a $VP([[]])$.

The second step is to change the rule for $S$ to say that it requires a verb phrase that has
all its complements and thus has the subcat list $[]$. This means that “I grab the gold” is a legal
sentence, but “You give” is not. The new rule,

$$S \rightarrow NP(Subjective) \hspace{0.5cm} VP([]),$$

can be read as “A sentence can be composed of a $NP$ in the subjective case, followed by a
$VP$ that has a null subcat list.” Figure 22.13 shows a parse tree using this grammar.

The third step is to remember that, in addition to complements, verb phrases (and other
phrases) can also take adjuncts, which are phrases that are not licensed by the individual verb
but rather may appear in any verb phrase. Phrases representing time and place are adjuncts,
because almost any action or event can have a time or place. For example, the adverb “now”
in “I smell a wumpus now” and the $PP$ “on Tuesday” in “give me the gold on Tuesday” are
adjuncts. Here are two rules that allow propositional and adverbial adjuncts on any $VP$:

$$VP(subcat) \rightarrow \begin{array}{l}
\text{VP}(subcat) \hspace{0.5cm} PP \\
| \hspace{1cm} \text{VP}(subcat) \hspace{0.5cm} Adverb.
\end{array}$$

### Generative capacity of augmented grammars

Each augmented rule is a rule schema, that stands for a set of rules, one for each possible
combination of values for the augmented constituents. The generative capacity of augmented
grammars depends on the number of combinations. If there is a finite number, then the
augmented grammar is equivalent to a context-free grammar: the rule schema could be re-
placed with individual context-free rules. But if there are an infinite number of values, then
augmented grammars can represent non-context-free languages. For example, the context-
sensitive language $a^nb^nc^n$ can be represented as:

$$S(n) \rightarrow A(n) \hspace{0.5cm} B(n) \hspace{0.5cm} C(n)$$

$$A(1) \rightarrow a \hspace{0.5cm} A(n+1) \rightarrow a \hspace{0.5cm} A(n)$$

$$B(1) \rightarrow b \hspace{0.5cm} B(n+1) \rightarrow b \hspace{0.5cm} B(n)$$

$$C(1) \rightarrow c \hspace{0.5cm} C(n+1) \rightarrow c \hspace{0.5cm} C(n)$$
22.5 **Semantic Interpretation**

So far, we have only looked at the syntactic analysis of language. In this section, we turn to the **semantics**—the extraction of the *meaning* of utterances. For this chapter we are using first-order logic as our representation language, so semantic interpretation is the process of associating an FOL expression with a phrase. Intuitively, the meaning of the phrase "the wumpus" is the big, hairy beast that we represent in logic as the logical term *Wumpus*$_1$, and the meaning of "the wumpus is dead" is the logical sentence *Dead*(Wumpus$_1$). This section will make that intuition more precise. We'll start with a simple example: a rule for describing grid locations:

$$NP \rightarrow Digit\ Digit.$$ 

We will augment the rule by adding to each constituent an argument representing the semantics of the constituent. We get

$$NP([x, y]) \rightarrow Digit(x)\ Digit(y).$$

This says that a string consisting of a digit with semantics $x$ followed by another digit with semantics $y$ forms an $NP$ with semantics $[x, y]$, which is our notation for a square in the grid.

Notice that the semantics of the whole $NP$ is composed largely of the semantics of the constituent parts. We have seen this idea of **compositional semantics** before: in logic, the meaning of $P \land Q$ is determined by the meaning of $P$, $Q$, and $\land$; in arithmetic, the meaning of $x + y$ is determined by the meaning of $x$, $y$, and $+$. Figure 22.14 shows how DCG notation can be used to augment a grammar for arithmetic expressions with semantics and Figure 22.15
Figure 22.14  A grammar for arithmetic expressions, augmented with semantics. Each variable \( x_i \) represents the semantics of a constituent. Note the use of the \{test\} notation to define logical predicates that must be satisfied, but that are not constituents.

Figure 22.15  Parse tree with semantic interpretations for the string “3 + (4 ÷ 2)”.

shows the parse tree for 3 + (4 ÷ 2) according to this grammar. The root of the parse tree is \( \text{Exp}(5) \), an expression whose semantic interpretation is 5.

The semantics of an English fragment

We are now ready to write the semantic augmentations for a fragment of English. We start by determining what semantic representations we want to associate with what phrases. We will use the simple example sentence “John loves Mary.” The NP “John” should have as its semantic interpretation the logical term \( \text{John} \), and the sentence as a whole should have as its interpretation the logical sentence \( \text{loves}(\text{John}, \text{Mary}) \). That much seems clear. The complicated part is the VP “loves Mary.” The semantic interpretation of this phrase is neither a logical term nor a complete logical sentence. Intuitively, “loves Mary” is a description that
might or might not apply to a particular person. (In this case, it applies to John.) This means that “loves Mary” is a predicate that, when combined with a term that represents a person (the person doing the loving), yields a complete logical sentence. Using the \( \lambda \)-notation (see page 248), we can represent “loves Mary” as the predicate
\[
\lambda x \text{ Loves}(x, \text{Mary})
\]
Now we need a rule that says “an NP with semantics \( \text{obj} \) followed by a VP with semantics \( \text{rel} \) yields a sentence whose semantics is the result of applying \( \text{rel} \) to \( \text{obj} \):)”
\[
S(\text{rel}(\text{obj})) \rightarrow \text{NP}(\text{obj}) \ \text{VP}(\text{rel})
\]
The rule tells us that the semantic interpretation of “John loves Mary” is
\[
(\lambda x \text{ Loves}(x, \text{Mary}))(\text{John})
\]
which is equivalent to \( \text{Loves}(\text{John}, \text{Mary}) \).

The rest of the semantics follows in a straightforward way from the choices we have made so far. Because \( \text{VPs} \) are represented as predicates, it is a good idea to be consistent and represent verbs as predicates as well. The verb “loves” is represented as \( \lambda y \lambda x \text{ Loves}(x, y) \), the predicate that, when given the argument \( \text{Mary} \), returns the predicate \( \lambda x \text{ Loves}(x, \text{Mary}) \).

The \( \text{VP} \rightarrow \text{Verb} \text{ NP} \) rule applies the predicate that is the semantic interpretation of the verb to the object that is the semantic interpretation of the \( \text{NP} \) to get the semantic interpretation of the whole \( \text{VP} \). We end up with the grammar shown in Figure 22.16 and the parse tree shown in Figure 22.17.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S(\text{rel}(\text{obj})) \rightarrow \text{NP}(\text{obj}) \ \text{VP}(\text{rel}) )</td>
<td></td>
</tr>
<tr>
<td>( \text{VP}(\text{rel}(\text{obj})) \rightarrow \text{Verb}(\text{rel}) \ \text{NP}(\text{obj}) )</td>
<td></td>
</tr>
<tr>
<td>( \text{NP}(\text{obj}) \rightarrow \text{Name}(\text{obj}) )</td>
<td></td>
</tr>
<tr>
<td>( \text{Name}(\text{John}) \rightarrow \text{John} )</td>
<td></td>
</tr>
<tr>
<td>( \text{Name}(\text{Mary}) \rightarrow \text{Mary} )</td>
<td></td>
</tr>
<tr>
<td>( \text{Verb}(\lambda y \lambda x \text{ Loves}(x, y)) \rightarrow \text{loves} )</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 22.16** A grammar that can derive a parse tree and semantic interpretation for “John loves Mary” (and three other sentences). Each category is augmented with a single argument representing the semantics.

**Time and tense**

Now suppose we want to represent the difference between “John loves Mary” and “John loved Mary.” English uses verb tenses (past, present, and future) to indicate the relative time of an event. One good choice to represent the time of events is the event calculus notation of Section 10.3. In event calculus, our two sentences have the following interpretations:

\[
e \in \text{Loves}(\text{John}, \text{Mary}) \land \text{During}(\text{Now}, e); \\
e \in \text{Loves}(\text{John}, \text{Mary}) \land \text{After}(\text{Now}, e).
\]
Section 22.5. Semantic Interpretation

This suggests that our two lexical rules for the words "loves" and "loved" should be these:

\[ \text{Verb}(\lambda y \lambda x \, x \in \text{Loves}(\text{John}, \text{Mary}) \land \text{During}(\text{Now}, e)) \rightarrow \text{loves} ; \]
\[ \text{Verb}(\lambda y \lambda x \, x \in \text{Loves}(x, y) \land \text{After}(\text{Now}, e)) \rightarrow \text{loved} . \]

Other than this change, everything else about the grammar remains the same, which is encouraging news; it suggests we are on the right track if we can so easily add a complication like the tense of verbs (although we have just scratched the surface of a complete grammar for time and tense). With this success as a warm-up, we are now ready to tackle a much harder representation problem.

Quantification

Consider the sentence "Every agent smells a wumpus." The sentence is actually ambiguous; the preferred meaning is that the agents might be smelling different wumpuses, but an alternative meaning is that there is a single wumpus that everyone smells.\(^{10}\) The two interpretations can be represented as follows:

\[ \forall a \, a \in \text{Agents} \implies \exists w \, w \in \text{Wumpuses} \land \exists e \, e \in \text{Smell}(a, w) \land \text{During}(\text{Now}, e) ; \]
\[ \exists w \, w \in \text{Wumpuses} \forall a \, a \in \text{Agents} \implies \exists e \, e \in \text{Smell}(a, w) \land \text{During}(\text{Now}, e) . \]

We will defer the problem of ambiguity and for now look only at the first interpretation. We'll try to analyze it compositionally, breaking it into the NP and VP components:

\[ \text{Every agent} \quad \text{NP}(\forall a \, a \in \text{Agents} \implies P) \]
\[ \text{smells a wumpus} \quad \text{VP}(\exists w \, w \in \text{Wumpuses} \land \exists e \, e \in \text{Smell}(a, w) \land \text{During}(\text{Now}, e)) . \]

Right away, there are two difficulties. First, the semantics of the entire sentence appears to be the semantics of the NP, with the semantics of the VP filling in the P part. That means that we cannot form the semantics of the sentence with \text{rel}(\text{obj}). We could do it with \text{obj}(\text{rel}), which seems a little odd (at least at first glance). The second problem is that we need to get

\(^{10}\) If this interpretation seems unlikely, consider "Every Protestant believes in a just God."
the variable $a$ as an argument of the relation $Smell$. In other words, the semantics of the sentence is formed by plugging the semantics of the $VP$ into the correct argument slot of the $NP$, while also plugging the variable $a$ from the $NP$ into the correct argument slot of the semantics of the $VP$. It looks as if we need two functional compositions and promises to be rather confusing. The complexity stems from the fact that the semantic structure is very different from the syntactic structure.

To avoid this confusion, many modern grammars take a different tack. They define an **intermediate form** to mediate between syntax and semantics. The intermediate form has two key properties. First, it is structurally similar to the syntax of the sentence and thus can be easily constructed through compositional means. Second, it contains enough information that it can be translated into a regular first-order logical sentence. Because it sits between the syntactic and logical forms, it is called a **quasi-logical form**. In this section, we will use a quasi-logical form that includes all of first-order logic and is augmented by lambda expressions and one new construction, which we will call a **quantified term**. The quantified term that is the semantic interpretation of "every agent" is written

$$[orall a a \in Agents].$$

This looks like a logical sentence, but it is used in the same way that a logical term is used. The interpretation of "Every agent smells a wumpus" in quasi-logical form is

$$\exists e \ (e \in Smell([\forall a a \in Agents], [\exists w w \in Wumpuses]) \land During(\text{Now}, e)).$$

To generate quasi-logical form, many of our rules remain unchanged. The rule for $S$ still creates the semantics of the $S$ with $\text{rel}(\text{obj})$. Some rules do change; the lexical rule for "a" is

$$\text{Article}(\exists) \rightarrow a$$

and the rule for combining an article with a noun is

$$NP([q \ x \ \text{sem}(x)]) \rightarrow \text{Article}(q) \ \text{Noun}(\text{sem}).$$

This says that the semantics of the $NP$ is a quantified term, with a quantifier specified by the article, with a new variable $x$, and with a proposition formed by applying the semantics of the noun to the variable $x$. The other rules for $NP$ are similar. Figure 22.18 shows the semantic types and example forms for each syntactic category under the quasi-logical form approach. Figure 22.19 shows the parse of "every agent smells a wumpus" using this approach, and Figure 22.20 shows the complete grammar.

Now we need to convert the quasi-logical form into real first-order logic by turning quantified terms into real terms. This is done by a simple rule: For each quantified term $[q \ x \ P(x)]$ within a quasi-logical form $QLF$, replace the quantified term with $x$, and replace $QLF$ with $q \ x \ P(x)$ $op$ $QLF$, where $op$ is $\rightarrow$ when $q$ is $\forall$ and is $\land$ when $q$ is $\exists$ or $\exists$. For example, the sentence "Every dog has a day" has the quasi-logical form:

$$\exists e \ (e \in Has([\forall d d \in Dogs], [\exists a a \in Days], \text{Now})).$$

---

11 Some quasi-logical forms have the third property that they can succinctly represent ambiguities that could be represented in logical form only by a long disjunction.
### Table 22.18

<table>
<thead>
<tr>
<th>Category</th>
<th>Semantic Type</th>
<th>Example</th>
<th>Quasi-Logical Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>sentence</td>
<td>I sleep.</td>
<td>[∃e ; e ∈ Sleep(Speaker) ∧ During(Now, e)]</td>
</tr>
<tr>
<td>NP</td>
<td>object</td>
<td>a dog</td>
<td>[∃d ; Dog(d)]</td>
</tr>
<tr>
<td>PP</td>
<td>object (\rightarrow) sentence</td>
<td>in [2,2]</td>
<td>[λx ; In(x, [2,2])]</td>
</tr>
<tr>
<td>RelClause</td>
<td>object (\rightarrow) sentence</td>
<td>that sees me</td>
<td>[λx ; ∃e ; e ∈ Sees(x, Speaker) ∧ During(Now, e)]</td>
</tr>
<tr>
<td>VP</td>
<td>object (^n) (\rightarrow) sentence</td>
<td>sees me</td>
<td>[λx ; ∃e ; e ∈ Sees(x, Speaker) ∧ During(Now, e)]</td>
</tr>
<tr>
<td>Adjective</td>
<td>object (\rightarrow) sentence</td>
<td>smelly</td>
<td>[λx ; Smelly(x)]</td>
</tr>
<tr>
<td>Adverb</td>
<td>event (\rightarrow) sentence</td>
<td>today</td>
<td>[λe ; During(e, Today)]</td>
</tr>
<tr>
<td>Article</td>
<td>quantifier</td>
<td>the</td>
<td>[∃! ]</td>
</tr>
<tr>
<td>Conjunction</td>
<td>sentence (^2) (\rightarrow) sentence</td>
<td>and</td>
<td>[λp, q ; (p ∧ q)]</td>
</tr>
<tr>
<td>Digit</td>
<td>object</td>
<td>7</td>
<td>[7]</td>
</tr>
<tr>
<td>Noun</td>
<td>object (\rightarrow) sentence</td>
<td>wumpus</td>
<td>[λx ; x ∈ Wumpuses]</td>
</tr>
<tr>
<td>Preposition</td>
<td>object (^2) (\rightarrow) sentence</td>
<td>in</td>
<td>[λx ; λy ; In(x, y)]</td>
</tr>
<tr>
<td>Pronoun</td>
<td>object</td>
<td>1</td>
<td>[Speaker]</td>
</tr>
<tr>
<td>Verb</td>
<td>object (^n) (\rightarrow) sentence</td>
<td>eats</td>
<td>[λy ; λx ; ∃e ; e ∈ Eats(x, y) ∧ During(Now, e)]</td>
</tr>
</tbody>
</table>

We did not specify which of the two quantified terms gets pulled out first, so there are actually two possible logical interpretations:

\[
∀d \; d ∈ Dogs ⇒ ∃a \; a ∈ Days ∧ ∃e \; e ∈ Has\(d, a, Now\) ;
\]
\[
∃a \; a ∈ Days ∧ ∀d \; d ∈ Dogs ⇒ ∃e \; e ∈ Has\(d, a, Now\) .
\]

The first one says that each dog has his own day, while the second says that there is a special day that all dogs share. Choosing between them is a job for disambiguation. Often, the left-to-right order of the quantified terms matches the left-to-right order of the quantifiers, but other factors come into play. The advantage of quasi-logical form is that it succinctly represents all the possibilities. The disadvantage is that it doesn’t help you choose between them; for that we need the full power of disambiguation using all sources of evidence.

### Pragmatic Interpretation

We have shown how an agent can perceive a string of words and use a grammar to derive a set of possible semantic interpretations. Now we address the problem of completing the interpretation by adding context-dependent information about the current situation to each candidate interpretation.
Figure 22.19  Parse tree for the sentence “Every agent smells a wumpus,” showing both syntactic structure and semantic interpretations.

Figure 22.20  A grammar with semantics in quasi-logical form.
Section 22.5. Semantic Interpretation

The most obvious need for pragmatic information is in resolving the meaning of indexicals, which are phrases that refer directly to the current situation. For example, in the sentence “I am in Boston today,” the interpretations of the indexicals “I” and “today” depend on who uttered the sentence when. We represent indexicals by “constants” (such as Speaker) which actually are fluents—that is, they depend on the situation. The hearer who perceives a speech act should also perceive who the speaker is and use this information to resolve the indexical. For example, the hearer might know $T((\text{Speaker} = \text{Agent}_B), \text{Now})$.

A command such as “go to 2 2” implicitly refers to the hearer. So far, our grammar for $S$ covers only declarative sentences. We can easily extend it to cover commands.\(^{12}\)

A command can be formed from a $VP$, where the subject is implicitly the hearer. We need to distinguish commands from statements, so we alter the rules for $S$ to include the type of speech act as part of the quasi-logical form:

\[
S(\text{Statement}(\text{Speaker}, \text{rel}(\text{obj}))) \rightarrow \text{NP}(\text{obj}) \text{} \text{VP}(\text{rel}) \\
S(\text{Command}(\text{Speaker}, \text{rel}(\text{Hearer}))) \rightarrow \text{VP}(\text{rel}).
\]

So the quasi-logical form for “Go to 2 2” is\(^{13}\)

\[
\text{Command}(\exists e \in \text{Go}(\text{Hearer}, [2, 2])).
\]

Language generation with DCGs

So far, we have concentrated on parsing language, not on generating it. Generation is a topic of similar richness. Choosing the right utterance to express a proposition involves many of the same choices that parsing the utterance does.

Remember that a DCG is a logical programming system that specifies constraints between a string and the parse of a string. We know that a logic programming definition of the $\text{Append}$ predicate can be used both to tell us that in $\text{Append}([1, 2], [3], x)$ we have $x = [1, 2, 3]$ and to enumerate the values of $x$ and $y$ that make $\text{Append}(x, y, [1, 2, 3])$ true. In the same way, we can write a definition of $S$ that can be used in two ways: to parse, we ask $S(\text{sem}, [\text{John}, \text{Loves}, \text{Mary}])$ and get back $\text{sem} = \text{Loves}(\text{John}, \text{Mary})$; to generate, we ask $S(\text{Loves}(\text{John}, \text{Mary}), \text{words})$ and get back $\text{words} = [\text{John}, \text{Loves}, \text{Mary}]$. We can also test a grammar by asking $S(\text{sem}, \text{words})$ and getting back as an answer a stream of $[\text{sem}, \text{words}]$ pairs that are generated by the grammar.

This approach works for the simple grammars in this chapter, but there can be difficulties in scaling up to larger grammars. The search strategy used by the logical inference engine is important; depth-first strategies can lead to infinite loops. Some care must be taken in the

---

\(^{12}\) To implement a complete communicating agent we would also need a grammar of questions. Questions are beyond the scope of this book because they impose long-distance dependencies between constituents. For example, in “Whom did the agent tell you to give the gold to?” the final word “to” should be parsed as a $PP$ with a missing $NP$; the missing $NP$ is licensed by the first word of the sentence, “whom.” A complex system of augmentations is used to make sure that the missing $NP$s match up with the licensing words.

\(^{13}\) Note that the quasi-logical form for a command does not include the time of the event (e.g., $\text{During}(\text{Now}, e)$). That is because the “go” is actually the untensed version of the word, not the present tense version. You can’t tell the difference with “go,” but observe that the correct form of a command is “Be good!” (using the untensed form “be”), not “Are good!” To ensure the correct tense is used, we could augment $VP$s with a tense argument and write $VP(\text{rel}, \text{untensed})$ on the right-hand side of the command rule.
exact details of the semantic form. It could be that a given grammar has no way to express the logical form \( X \land Y \) for some values of \( X \) and \( Y \); but can express \( Y \land X \); this suggests that we need a way to canonicalize semantic forms, or we need to extend the unification routine so that \( X \land Y \) can unify with \( Y \land X \).

Serious work in generation tends to use more complex generation models that are distinct from the parsing grammar and offer more control over exactly how components of the semantics are expressed. Systemic grammar is one approach that makes it easy to put emphasis on the most important parts of the semantic form.

### 22.6 Ambiguity and Disambiguation

In some cases, hearers are consciously aware of ambiguity in an utterance. Here are some examples taken from newspaper headlines:

- Squad helps dog bite victim.
- Helicopter powered by human flies.
- Once-sagging cloth diaper industry saved by full dumps.
- Portable toilet bombed; police have nothing to go on.
- British left waffles on Falkland Islands.
- Teacher strikes idle kids.
- Milk drinkers are turning to powder.
- Drunk gets nine months in violin case.

But most of the time the language we hear seems unambiguous. Thus, when researchers first began to use computers to analyze language in the 1960s they were quite surprised to learn that almost every utterance is highly ambiguous, even though the alternative interpretations might not be apparent to a native speaker. A system with a large grammar and lexicon might find thousands of interpretations for a perfectly ordinary sentence. Consider “The batter hit the ball,” which seems to have an unambiguous interpretation in which a baseball player strikes a baseball. But we get a different interpretation if the previous sentence is “The mad scientist unleashed a tidal wave of cake mix towards the ballroom.” This example relies on **lexical ambiguity**, in which a word has more than one meaning. Lexical ambiguity is quite common; “back” can be an adverb (go back), an adjective (back door), a noun (the back of the room) or a verb (back up your files). “Jack” can be a name, a noun (a playing card, a six-pointed metal game piece, a nautical flag, a fish, a male donkey, a socket, or a device for raising heavy objects), or a verb (to jack up a car, to hunt with a light, or to hit a baseball hard).

**Syntactic ambiguity** (also known as **structural ambiguity**) can occur with or without lexical ambiguity. For example, the string “I smelled a wumpus in 2,2” has two parses: one where the prepositional phrase “in 2,2” modifies the noun and one where it modifies the verb. The syntactic ambiguity leads to a **semantic ambiguity**, because one parse means that the wumpus is in 2,2 and the other means that a stench is in 2,2. In this case, getting the wrong interpretation could be a deadly mistake.
Semantic ambiguity can occur even in phrases with no lexical or syntactic ambiguity. For example, the noun phrase “cat person” can be someone who likes felines or the lead of the movie *Attack of the Cat People*. A “coast road” can be a road that follows the coast or one that leads to it.

Finally, there can be ambiguity between literal and figurative meanings. Figures of speech are important in poetry, but are surprisingly common in everyday speech as well. A **metonymy** is a figure of speech in which one object is used to stand for another. When we hear “Chrysler announced a new model,” we do not interpret it as saying that companies can talk; rather we understand that a spokesperson representing the company made the announcement. Metonymy is common and is often interpreted unconsciously by human hearers. Unfortunately, our grammar as it is written is not so facile. To handle the semantics of metonymy properly, we need to introduce a whole new level of ambiguity. We do this by providing two objects for the semantic interpretation of every phrase in the sentence: one for the object that the phrase literally refers to (Chrysler) and one for the metonymic reference (the spokesperson). We then have to say that there is a relation between the two. In our current grammar, “Chrysler announced” gets interpreted as

$$\exists x, e \ x = Chrysler \land e \in Announce(x) \land After(Now, e).$$

We need to change that to

$$\exists m, x, e \ x = Chrysler \land e \in Announce(m) \land After(Now, e) \land Metonymy(m, x).$$

This says that there is one entity $x$ that is equal to Chrysler, and another entity $m$ that did the announcing, and that the two are in a metonymy relation. The next step is to define what kinds of metonymy relations can occur. The simplest case is when there is no metonymy at all—the literal object $x$ and the metonymic object $m$ are identical:

$$\forall m, x \ (m = x) \Rightarrow Metonymy(m, x).$$

For the Chrysler example, a reasonable generalization is that an organization can be used to stand for a spokesperson of that organization:

$$\forall m, x \ x \in Organizations \land Spokesperson(m, x) \Rightarrow Metonymy(m, x).$$

Other metonymies include the author for the works (I read *Shakespeare*) or more generally the producer for the product (I drive a *Honda*) and the part for the whole (The Red Sox need a strong *arm*). Some examples of metonymy, such as “The *ham sandwich* on Table 4 wants another beer,” are more novel and are interpreted with respect to a situation.

The rules we have outlined here allow us to construct an explanation for “Chrysler announced a new model,” but the explanation doesn’t follow by logical deduction. We need to use probabilistic or nonmonotonic reasoning to come up with candidate explanations.

A **metaphor** is a figure of speech in which a phrase with one literal meaning is used to suggest a different meaning by way of analogy. Most people think of metaphor as a tool used by poets that does not play a large role in everyday text. However, a number of basic metaphors are so common that we do not even recognize them as such. One such metaphor is the idea that *more is up*. This metaphor allows us to say that prices have risen, climbed, or
skyrocketed, that the temperature has dipped or fallen, that one’s confidence has plummeted, or that a celebrity’s popularity has jumped or soared.  

There are two ways to approach metaphors like this. One is to compile all knowledge of the metaphor into the lexicon—to add new senses of the words “rise,” “fall,” “climb,” and so on, that describe them as dealing with quantities on any scale rather than just altitude. This approach suffices for many applications, but it does not capture the generative character of the metaphor that allows humans to use new instances such as “nosedive” or “blasting through the roof” without fear of misunderstanding. The second approach is to include explicit knowledge of common metaphors and use them to interpret new uses as they are read. For example, suppose the system knows the “more is up” metaphor. That is, it knows that logical expressions that refer to a point on a vertical scale can be interpreted as being about corresponding points on a quantity scale. Then the expression “sales are high” would get a literal interpretation along the lines of \( Altitude(Sales, \text{High}) \), which could be interpreted metaphorically as \( Quantity(Sales, \text{Much}) \).

**Disambiguation**

As we said before, disambiguation is a question of diagnosis. The speaker’s intent to communicate is an unobserved cause of the words in the utterance, and the hearer’s job is to work backwards from the words and from knowledge of the situation to recover the most likely intent of the speaker. In other words, the hearer is solving for

\[
\arg\max_{\text{intent}} \text{Likelihood(intent|words, situation)},
\]

where \( \text{Likelihood} \) can either be probability or any numeric measure of preference. Some sort of preference is needed because syntactic and semantic interpretation rules alone cannot identify a unique correct interpretation of a phrase or sentence. So we divide the work: syntactic and semantic interpretation is responsible for enumerating a set of candidate interpretations, and the disambiguation process chooses the best one.

Note that we talk about the intent of the speech act, not just the actual proposition that the speaker is proclaiming. For example, after hearing a politician say, “I am not a crook,” we might assign a probability of only 50% to the proposition that the politician is not a criminal, and 99.9999% to the proposition that the speaker is not a hooked shepherd’s staff. Still, we assign a higher probability to the interpretation

\[
\text{Assert}(\text{Speaker}, \neg(\text{Speaker} \in \text{Criminals}))
\]

because this is a more likely thing to say.

Consider again the ambiguous example “I smelled a wumpus in 2,2.” One preference heuristic is the rule of **right association**, which says that when it is time to decide where in the parse tree to place the PP “in 2,2,” we should prefer to attach it to the rightmost existing constituent, which in this case is the NP “wumpus.” Of course, this is only a heuristic; for the sentence “I smelled a wumpus with my nose,” the heuristic would be outweighed by the fact that the NP “a wumpus with my nose” is unlikely.

Disambiguation is made possible by combining evidence, using all the techniques for knowledge representation and uncertain reasoning that we have seen throughout this book.
We can break the knowledge down into four models:

1. The **world model**: the likelihood that a proposition occurs in the world.

2. The **mental model**: the likelihood that the speaker forms the intention of communicating a certain fact to the hearer, given that it occurs. This approach combines models of what the speaker believes, what the speaker believes the hearer believes, and so on.

3. The **language model**: the likelihood that a certain string of words will be chosen, given that the speaker has the intention of communicating a certain fact. The CFG and DCG models presented in this chapter have a Boolean model of likelihood: either a string can have a certain interpretation or it cannot. In the next chapter, we will see a probabilistic version of CFG that makes for a more informed language model for disambiguation.

4. The **acoustic model**: the likelihood that a particular sequence of sounds will be generated, given that the speaker has chosen a given string of words. Section 15.6 covered speech recognition.

### 22.7 Discourse Understanding

A discourse is any string of language—usually one that is more than one sentence long. Textbooks, novels, weather reports and conversations are all discourses. So far we have largely ignored the problems of discourse, preferring to dissect language into individual sentences that can be studied in vitro. This section studies sentences in their native habitat. We will look at two particular subproblems: reference resolution and coherence.

#### Reference resolution

Reference resolution is the interpretation of a pronoun or a definite noun phrase that refers to an object in the world.\(^{14}\) The resolution is based on knowledge of the world and of the previous parts of the discourse. Consider the passage:

"John flagged down the waiter. He ordered a ham sandwich."

To understand that "he" in the second sentence refers to John, we need to have understood that the first sentence mentions two people and that John is playing the role of a customer and hence is likely to order, whereas the waiter is not. Usually, reference resolution is a matter of selecting a referent from a list of candidates, but sometimes it involves the creation of new candidates. Consider the following sentence:

"After John proposed to Marsha, they found a preacher and got married. For the honeymoon, they went to Hawaii."

Here, the definite noun phrase "the honeymoon" refers to something that was only implicitly alluded to by the verb "married." The pronoun "they" refers to a group that was not explicitly mentioned before: John and Marsha (but not the preacher).

---

\(^{14}\) In linguistics, reference to something that has already been introduced is called *anaphoric* reference. Reference to something yet to be introduced is called *cataphoric* reference, as with the pronoun "he" in "When he won his first tournament, Tiger was 20."
Choosing the best referent is a process of disambiguation that relies on combining a variety of syntactic, semantic, and pragmatic information. Some clues are in the form of constraints. For example, pronouns must agree in gender and number with their antecedents: "he" can refer to John, but not Marsha; "they" can refer to a group, but not a single person. Pronouns must also obey syntactic constraints for reflexivity. For example, in "He saw him in the mirror" the two pronouns must refer to different people, whereas in "He saw himself," they must refer to the same person. There are also constraints for semantic consistency. In "He ate it," the pronoun "he" must refer to something that eats and "it" to something that can be eaten.

Some clues are preferences that do not always hold. For example, when adjacent sentences have a parallel structure, it is preferable for pronominal reference to follow that structure. So in

Marsha flew to San Francisco from New York. John flew there from Boston.

we prefer for "there" to refer to San Francisco because it plays the same syntactic role. Absent a parallel structure, there is a preference for subjects over objects as antecedents. Thus, in

Marsha gave Sally the homework assignment. Then she left.

"Marsha," the subject of the first sentence, is the preferred antecedent for "she." Another preference is for the entity that has been discussed most prominently. Considered in isolation, the pair of sentences

Dana dropped the cup on the plate. It broke.

poses a problem: it is not clear whether the cup or the plate is the referent of "it." But in a larger context the ambiguity is resolved:

Dana was quite fond of the blue cup. The cup had been a present from a close friend. Unfortunately, one day while setting the table. Dana dropped the cup on the plate. It broke.

Here, the cup is the focus of attention and hence is the preferred referent.

A variety of reference resolution algorithms have been devised. One of the first (Hobbs, 1978) is remarkable because it underwent a degree of statistical verification that was unusual for the time. Using three different genres of text, Hobbs reports an accuracy of 92%. This assumed that a correct parse was generated by a parser; not having one available, Hobbs constructed the parses by hand. The Hobbs algorithm works as a search: it searches sentences starting from the current sentence and going backwards. This technique ensures that more recent candidates will be considered first. Within a sentence it searches breadth first, from left to right. This ensures that subjects will be considered before objects. The algorithm chooses the first candidate that satisfies the constraints just outlined.

The structure of coherent discourse

Open up this book to 10 random pages, and copy down the first sentence from each page. The result is bound to be incoherent. Similarly, if you take a coherent 10-sentence passage and permute the sentences, the result is incoherent. This demonstrates that sentences in natural
- **Enable or cause**: $S_1$ brings about a change of state (which may be implicit) that causes $S_2$. Example: "I went outside. I drove to school." (Going outside enables the implicit getting into a car.)
- **Explanation**: The reverse of enablement: $S_2$ causes or enables $S_1$ and thus is an explanation for it. Example: "I was late for school. I overslept."
- **Ground-Figure**: $S_1$ describes a setting or background for $S_2$. Example: "It was a dark and stormy night. Rest of story."
- **Evaluation**: From $S_2$ infer that $S_1$ is part of the speaker's plan for executing the segment as a speech act. Example: "A funny thing happened. Rest of story."
- **Exemplification**: $S_2$ is an example of the general principle in $S_1$. Example: "This algorithm reverses a list. The input $[A, B, C]$ is mapped to $[C, B, A]$."
- **Generalization**: $S_1$ is an example of the general principle in $S_2$. Example: "[$A, B, C$] is mapped to $[C, B, A]$. In general, the algorithm reverses a list."
- **Violated Expectation**: Infer $\neg P$ from $S_2$, negating the normal inference of $P$ from $S_1$. Example: "This paper is weak. On the other hand, it is interesting."

Figure 22.21 A list of coherence relations, taken from Hobbs (1990). Each relation holds between two adjacent text segments, $S_1$ and $S_2$.  

Language discourse are quite different from sentences in logic. In logic, if we TELL sentences $A$, $B$ and $C$ to a knowledge base, in any order, we end up with the conjunction $A \land B \land C$. In natural language, sentence order matters; consider the difference between "Go two blocks. Turn right." and "Turn right. Go two blocks."

A discourse has structure above the level of a sentence. We can examine this structure with the help of a grammar of discourse:

\[
\begin{align*}
    \text{Segment}(x) & \rightarrow S(x) \\
    \text{Segment}(\text{CoherenceRelation}(x, y)) & \rightarrow \text{Segment}(x) \text{ Segment}(y) .
\end{align*}
\]

This grammar says that a discourse is composed of segments, where each segment is either a sentence or a group of sentences and where segments are joined by coherence relations. In the text "Go two blocks. Turn right," the coherence relation is that the first sentence enables the second: the listener should turn right only after traveling two blocks. Different researchers have proposed different inventories of coherence relations; Figure 22.21 lists a representative set. Now consider the following story:

1. A funny thing happened yesterday.
2. John went to a fancy restaurant.
3. He ordered the duck.
4. The bill came to $50.
5. John got a shock when he realized he had no money.
6. He had left his wallet at home.
7. The waiter said it was all right to pay later.
8. He was very embarrassed by his forgetfulness.
Here, sentence (1) stands in the Evaluation relation to the rest of the discourse; (1) is the speaker’s metacomment on the discourse. Sentence (2) enables (3), and together the (2–3) pair cause (4), with the implicit intermediate state that John ate the duck. Now (2–4) serve as the ground for the rest of the discourse. Sentence (6) is an explanation of (5), and (5–6) enable (7). Note that this is an Enable and not a Cause, because the waiter might have had a different reaction. Together, (5–7) cause (8). Exercise 22.13 asks you to draw the parse tree for this discourse.

Coherence relations serve to bind a discourse together. They guide the speaker in deciding what to say and what to leave implicit, and they guide the hearer in recovering the speaker’s intent. Coherence relations can serve as a filter on the ambiguity of sentences: individually, the sentences might be ambiguous, but most of these ambiguous interpretations do not fit together into a coherent discourse.

So far we have looked at reference resolution and discourse structure separately. But the two are actually intertwined. The theory of Groosz and Sidner (1986), for example, accounts for where the speaker’s and hearer’s attention is focused during the discourse. Their theory includes a pushdown stack of focus spaces. Certain utterances cause the focus to shift by pushing or popping elements off the stack. For example, in the restaurant story, the sentence “John went to a fancy restaurant” pushes a new focus onto the stack. Within that focus, the speaker can use a definite NP to refer to “the waiter” (rather than “a waiter”). If the story continued with “John went home,” then the focus space would be popped from the stack and the discourse could no longer refer to the waiter with “the waiter” or “he.”

22.8 Grammar Induction

Grammar induction is the task of learning a grammar from data. It is an obvious task to attempt, given that it has proven to be so difficult to construct a grammar by hand and that billions of example utterances are available for free on the Internet. It is a difficult task because the space of possible grammars is infinite and because verifying that a given grammar generates a set of sentences is computationally expensive.

One interesting model is the SEQUITUR system (Nevill-Manning and Witten, 1997). It requires no input except a single text (which does not need to be predivided into sentences). It produces a grammar of a very specialized form: a grammar that generates only a single string, namely, the original text. Another way to look at this is that SEQUITUR learns just enough grammar to parse the text. Here is the bracketing it discovers for one sentence within a larger text of news stories:

[Most Labour] [sentiment [[would still] [favor the] abolition]] [[of [the House]] [of Lords]]

It has correctly picked out constituents such as the PP “of the House of Lords,” although it also goes against traditional analysis in, for example, grouping “the” with the preceding verb rather than the following noun.

SEQUITUR is based on the idea that a good grammar is a compact grammar. In particular, it enforces the following two constraints: (1) No pair of adjacent symbols should appear
more than once in the grammar. If the symbol pair \( A \ B \) appears on the right-hand side of several rules, then we should replace the pair with a new nonterminal that we will call \( C \) and add the rule \( C \rightarrow A \ B \). (2) Every rule should be used at least twice. If a nonterminal \( C \) appears only once in the grammar, then we should eliminate the rule for \( C \) and replace its single use with the rule's right-hand side. These two constraints are applied in a greedy search that scans the input text from left to right, incrementally building a grammar as it goes, and imposing the constraints as soon as possible. Figure 22.22 shows the algorithm in operation on the input text "abcdefabcde." The algorithm recovers an optimally compact grammar for the text.

<table>
<thead>
<tr>
<th>Input</th>
<th>Grammar</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( a )</td>
<td>( S \rightarrow a )</td>
<td></td>
</tr>
<tr>
<td>2 ( ab )</td>
<td>( S \rightarrow ab )</td>
<td></td>
</tr>
<tr>
<td>3 ( abc )</td>
<td>( S \rightarrow abc )</td>
<td></td>
</tr>
<tr>
<td>4 ( abed )</td>
<td>( S \rightarrow abed )</td>
<td></td>
</tr>
<tr>
<td>5 ( abed )</td>
<td>( S \rightarrow abed )</td>
<td></td>
</tr>
<tr>
<td>6 ( abed )</td>
<td>( S \rightarrow abed )</td>
<td>( bc ) twice</td>
</tr>
<tr>
<td></td>
<td>( S \rightarrow aAdA; A \rightarrow bc )</td>
<td></td>
</tr>
<tr>
<td>7 ( abed )</td>
<td>( S \rightarrow aAdA; A \rightarrow bc )</td>
<td></td>
</tr>
<tr>
<td>8 ( abed )</td>
<td>( S \rightarrow aAdA; A \rightarrow bc )</td>
<td></td>
</tr>
<tr>
<td>9 ( abed )</td>
<td>( S \rightarrow aAdA; A \rightarrow bc )</td>
<td>( bc ) twice</td>
</tr>
<tr>
<td></td>
<td>( S \rightarrow aAdA; A \rightarrow bc )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( S \rightarrow aAdA; A \rightarrow bc )</td>
<td></td>
</tr>
<tr>
<td>10 ( abed )</td>
<td>( S \rightarrow abed )</td>
<td>( Bd ) twice</td>
</tr>
<tr>
<td></td>
<td>( S \rightarrow aAdA; A \rightarrow bc )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( S \rightarrow aAdA; A \rightarrow bc )</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 22.22** A trace of SEQUITUR inducing a grammar for the input text "abcdefabcde." We start with a rule for \( S \) and add each symbol to the end of this rule in turn. After adding the sixth symbol, we have the first occurrence of a repeated pair: \( bc \). So we replace both occurrences of \( bc \) with the new nonterminal \( A \) and add the rule \( A \rightarrow bc \). After three more symbols are added, the ninth causes another repetition of \( bc \), so again we replace it with \( A \). This leads to two occurrences of \( aA \), so we replace them with a new nonterminal, \( B \). After adding the tenth and last terminal symbol we get two occurrences of \( Bd \), so we replace them with the new nonterminal \( C \). But now \( B \) appears only once, in the right-hand side of the \( C \) rule, so we replace \( B \) by its expansion, \( aA \).

In the next chapter, we will see other grammar induction algorithms that work with probabilistic context-free grammars. But now we turn to the problem of learning a grammar that is augmented with semantics. Since an augmented grammar is a Horn clause logic program, the techniques of inductive logic programming are appropriate. C H I L L (Zelle and Mooney, 1996) is an inductive logic programming (ILP) program that learns a grammar and a specialized parser for that grammar from examples. The target domain is natural language
database queries. The training examples consist of pairs of word strings and corresponding queries—for example;

\[
\text{What is the capital of the state with the largest population?} \\
\text{Answer(c, Capital(s, c) \land Largest(p, State(s) \land Population(s, p)))}
\]

CHILL's task is to learn a predicate \textit{Parse(words, query)} that is consistent with the examples and, hopefully, generalizes well to other examples. Applying ILP directly to learn this predicate results in poor performance: the induced parser has only about 20\% accuracy. Fortunately, ILP learners can improve by adding knowledge. In this case, most of the \textit{Parse} predicate was defined as a logic program, and CHILL's task was reduced to inducing the control rules that guide the parser to select one parse over another. With this additional background knowledge, CHILL achieves 70\% to 85\% accuracy on various database query tasks.

22.9 SUMMARY

Natural language understanding is one of the most important subfields of AI. It draws on ideas from philosophy and linguistics, as well as on techniques of logical and probabilistic knowledge representation and reasoning. Unlike other areas of AI, natural language understanding requires an empirical investigation of actual human behavior—which turns out to be complex and interesting.

- Agents send signals to each other to achieve certain purposes: to inform, to warn, to elicit help, to share knowledge, or to promise something. Sending a signal in this way is called a \textit{speech act}. Ultimately, all speech acts are an attempt to get another agent to believe something or do something.

- Language consists of conventional \textit{signs} that convey meaning. Many animals use signs in this sense. Humans appear to be the only animals that use \textit{grammar} to produce an unbounded variety of structured messages.

- Communication involves three steps by the speaker: the intention to convey an idea, the mental generation of words, and their physical synthesis. The hearer then has four steps: perception, analysis, disambiguation, and incorporation of the meaning. All language use is \textit{situated}, in the sense that the meaning of an utterance can depend on the situation in which it is produced.

- Formal language theory and \textit{phrase structure} grammars (and in particular, \textit{context-free} grammar) are useful tools for dealing with some aspects of natural language.

- Sentences in a context-free language can be parsed in \(O(n^2)\) time by a \textit{chart parser}.

- It is convenient to \textit{augment} a grammar to handle such problems as subject–verb agreement and pronoun case. \textit{Definite clause grammar} (DCG) is a formalism that allows for augmentations. With DCG, parsing and semantic interpretation (and even generation) can be done using logical inference.

- \textit{Semantic interpretation} can also be handled by an augmented grammar. A quasi-logical form can be a useful intermediate between syntactic trees and semantics.
• **Ambiguity** is a very important problem in natural language understanding; most sentences have many possible interpretations, but usually only one is appropriate. Disambiguation relies on knowledge about the world, about the current situation, and about language use.

• Most language exists in the context of multiple sentences, not just a single one. **Discourse** is the study of connected texts. We saw how to resolve pronominal references across sentences and how sentences are joined into coherent segments.

• **Grammar induction** can learn a grammar from examples, although there are limitations on how well the grammar will generalize.

---

**BIBLIOGRAPHICAL AND HISTORICAL NOTES**

The study of signs and symbols as elements of language was named **semiotics** by John Locke (1690), although it was not developed until the 20th century (Peirce, 1902; de Saussure, 1993). Recent overview texts include Eco's (1979) and Coblentz's (1997).

The idea of language as action stems from 20th-century linguistically oriented philosophy (Wittgenstein, 1953; Grice, 1957; Austin, 1962) and particularly from the book *Speech Acts* (Searle, 1969). A precursor to the idea of speech acts was Protagoras's (c. 430 B.C.) identification of four types of sentence: prayer, question, answer, and injunction. A plan-based model of speech acts was suggested first by Cohen and Perrault (1979). Connecting language to action by using plan recognition to understand stories was studied by Wilensky (1983). Cohen, Morgan, and Pollack (1990) collect more recent work in this area.

Like semantic networks, context-free grammars (also known as phrase structure grammars) are a reinvention of a technique first used by ancient Indian grammarians (especially Panini, c. 350 B.C.) studying Sanskrit (Ingerman, 1967). They were reinvented by Noam Chomsky (1956) for the analysis of English syntax and independently by John Backus for the analysis of Algol-58 syntax. Naur (1963) extended Backus's notation and is now credited (Backus, 1996) with the "N" in BNF, which originally stood for "Backus Normal Form." Knuth (1968) defined a kind of augmented grammar called **attribute grammar** that is useful for programming languages. Definite clause grammars were introduced by Colmerauer (1975) and developed and popularized by Pereira and Warren (1980). The Prolog programming language was invented by Alain Colmerauer specifically for the problem of parsing the French language. Colmerauer actually introduced a formalism called metamorphosis grammar that went beyond definite clauses, but DCG followed soon after.

There have been many attempts to write formal grammars of natural languages, both in "pure" linguistics and in computational linguistics. Machine-oriented grammars include those developed in the Linguistic String Project at New York University (Sager, 1981) and the XTAG project at the University of Pennsylvania (Doran et al., 1994). A good example of a modern DCG system is the Core Language Engine (Alshawi, 1992). There are several comprehensive but informal grammars of English (Jespersen, 1965; Quirk et al., 1985; McCawley, 1988; Huddleston and Pullum, 2002). Good textbooks on linguistics include Sag and...

Since the mid-1980s, there has been a trend toward putting more information in the lexicon and less in the grammar. Lexical-functional grammar, or LFG, (Bresnan, 1982) was the first major grammar formalism to be highly lexicalized. If we carry lexicalization to an extreme, we end up with categorial grammar, in which there can be as few as two grammar rules, or dependency grammar (Melčuk and Pécqueur, 1988), in which there are no phrases, only words. Sleator and Temperley (1993) describe a popular parser that uses dependency grammar. Tree-Adjoining Grammar, or TAG, (Joshi, 1985) is not strictly lexical, but it is gaining popularity in its lexicalized form (Schabes et al., 1988). Wordnet (Fellbaum, 2001) is a publicly-available dictionary of about 100,000 words and phrases, categorized into parts of speech and linked by semantic relations such as synonym, antonym, and part-of.

The first computerized parsing algorithms were demonstrated by Yngve (1955). Efficient algorithms were developed in the late 1960s, with a few twists since then (Kasami, 1965; Younger, 1967; Graham et al., 1980). Our chart parser is closest to Earley’s (1970). A good summary appears in the text on parsing and compiling by Aho and Ullman (1972). Maxwell and Kaplan (1993) show how chart parsing with augmentations can be made efficient in the average case. Church and Patil (1982) address the resolution of syntactic ambiguity.

Formal semantic interpretation of natural languages originates within philosophy and formal logic and is especially closely related to Alfred Tarski’s (1935) work on the semantics of formal languages. Bar-Hillel was the first to consider the problems of pragmatics and propose that they could be handled by formal logic. For example, he introduced C. S. Peirce’s (1902) term indexical into linguistics (Bar-Hillel, 1954). Richard Montague’s essay “English as a formal language” (1970) is a kind of manifesto for the logical analysis of language, but the book by Dowty et al. (1991) and the article by Lewis (1972) are more readable. A complete collection of Montague’s contributions has been edited by Thomason (1974). In artificial intelligence, the work of McAllester and Givan (1992) continues the Montagovian tradition, adding many new technical insights.

The idea of an intermediate or quasi-logical form to handle problems such as quantifier scoping goes back to Woods (1978) and is present in many recent systems (Alshawi, 1992; Hwang and Schubert, 1993).

The first NLP system to solve an actual task was probably the baseball question answering system (Green et al., 1961), which handled questions about a database of baseball statistics. Close after that was Woods’s (1973) LUNAR, which answered questions about the rocks brought back by the Apollo program. Roger Schank and his students built a series of programs (Schank and Abelson, 1977; Wilensky, 1978; Schank and Riesbeck, 1981; Dyer, 1983) that all had the task of understanding language. The emphasis, however, was less on language per se and more on representation and reasoning. The problems included representing stereotypical situations (Cullingford, 1981), describing human memory organization (Rieger, 1976; Kolodner, 1983), and understanding plans and goals (Wilensky, 1983).

Natural language generation was considered from the earliest days of machine translation in the 1950s, but it didn’t appear as a monolingual concern until the 1970s. The work
by Simmons and Slocum (1972) and Goldmar. (1975) are representative. PENMAN (Bateman et al., 1989) was one of the first full-scale generation systems, based on Systemic Grammar (Kasper, 1988). In the 1990s, two important public-domain generation systems, KPML (Bateman, 1997) and FUF (Elhadad, 1993), became available. Important books on generation include McKeown (1985), Hovy (1988), Patten (1988) and Reiter and Dale (2000).

Some of the earliest work on disambiguation was Wilks's (1975) theory of preference semantics, which tried to find interpretations that minimize the number of semantic anomalies. Hirst (1987) describes a system with similar aims that is closer to the compositional semantics described in this chapter. Hobbs et al. (1993) describes a quantitative framework for measuring the quality of a syntactic and semantic interpretation. Since then, it has become more common to use an explicitly Bayesian framework (Charniak and Goldman, 1992; Wu, 1993). In linguistics, optimality theory (Linguistics) (Kager, 1999) is based on the idea of building soft constraints into the grammar, giving a natural ranking to interpretations, rather than having the grammar generate all possibilities with equal rank. Norvig (1988) discusses the problems of considering multiple simultaneous interpretations, rather than settling for a single maximum likelihood interpretation. Literary critics (Empson, 1953; Hobbs, 1990) have been ambiguous about whether ambiguity is something to be resolved or cherished.


Our treatment of reference resolution follows Hobbs (1978). A more complex model by Lappin and Leass (1994) is based on a quantitative scoring mechanism. More recent work (Kehler, 1997; Ge et al., 1998) has used machine learning to tune the quantitative parameters. Two excellent surveys of reference resolution are the books by Hirst (1981) and Mitkov (2002).

In 1758, David Hume's *Enquiry Concerning the Human Understanding* argued that discourse is connected by "three principles of connexion among ideas, namely Resemblance, Contiguity in time or place, and Cause or Effect." So began a long history of trying to define coherence relations. Hobbs (1990) gives us the set used in this chapter; Mann and Thompson (1983) provide a more elaborate set that includes solutionhood, evidence, justification, motivation, reason, sequence, enablement, elaboration, restatement, condition, circumstance, cause, concession, background, and thesis–anathesis. That model evolved into rhetorical structure theory (RST), which is perhaps the most prominent theory today (Mann and Thompson, 1988). This chapter borrows some of the examples from the chapter in Jurafsky and Martin (2000) written by Andrew Kehler.


The first important result on grammar induction was a negative one: Gold (1967) showed that it is not possible to reliably learn a correct context-free grammar, given a set of
strings from that grammar. Essentially, the idea is that, given a set of strings $s_1, s_2 \ldots s_n$, the correct grammar could be all-inclusive ($S \rightarrow \text{word}^*$), or it could be a copy of the input ($S \rightarrow s_1 | s_2 | \ldots | s_n$), or anywhere in between. Prominent linguists, such as Chomsky (1957, 1980) and Pinker (1989, 2000), have used Gold's result to argue that there must be an innate universal grammar that all children have from birth. The so-called Poverty of the Stimulus argument is that children have no language inputs other than positive examples: their parents and peers produce mostly accurate examples of their language, and very rarely correct mistakes. Therefore, because Gold proved that learning a CFG from positive examples is impossible, the children must already "know" the grammar and be merely tuning some of its parameters of this innate grammar and learning vocabulary. While this argument continues to hold sway throughout much of Chomskian linguistics, it has been dismissed by some other linguists (Pullum, 1996; Elman et al., 1997) and most computer scientists. As early as 1969, Horning showed that it is possible to learn, in the sense of PAC learning, a probabilistic context-free grammar. Since then there have been many convincing empirical demonstrations of learning from positive examples alone, such as the ILP work of Mooney (1999) and Muggleton and De Raedt (1994) and the remarkable Ph.D. theses of Schütze (1995) and de Marchen (1996). It is possible to learn other grammar formalisms, such as regular languages (Ocmina and Garcia, 1992; Denis, 2001), and regular tree languages (Carrasco et al., 1998), and finite state automata (Parekh and Honavar, 2001).

The SEQUITUR system is due to Nevill-Manning and Witten (1997). Interestingly, they, as well as de Marchen, remark that their grammar induction schemes are also good compression schemes. This is in accordance with the principle of minimal description length encoding: a good grammar is a grammar that minimizes the sum of two lengths: the length of the grammar and the length of the parse tree of the text.


The Association for Computational Linguistics (ACL) holds regular conferences and publishes the journal Computational Linguistics. There is also an International Conference on Computational Linguistics (COLING). Readings in Natural Language Processing (Grosz et al., 1986) is an anthology containing many important early papers. Dale et al. (2000) emphasize practical tools for building NLP systems. The textbook by Jurafsky and Martin (2000) gives a comprehensive introduction to the field. Allen (1995) is a slightly older treatment. Pereira and Sheiber (1987) and Covington (1994) offer concise overviews of syntactic processing based on implementations in Prolog. The Encyclopedia of AI has many useful articles on the field; see especially the entries "Computational Linguistics" and "Natural Language Understanding."
EXERCISES

22.1 Read the following text once for understanding, and remember as much of it as you can. There will be a test later.

The procedure is actually quite simple. First you arrange things into different groups. Of course, one pile may be sufficient depending on how much there is to do. If you have to go somewhere else due to lack of facilities that is the next step, otherwise you are pretty well set. It is important not to overdo things. That is, it is better to do too few things at once than too many. In the short run this may not seem important but complications can easily arise. A mistake is expensive as well. At first the whole procedure will seem complicated. Soon, however, it will become just another facet of life. It is difficult to foresee any end to the necessity for this task in the immediate future, but then one can never tell. After the procedure is completed one arranges the material into different groups again. Then they can be put into their appropriate places. Eventually they will be used once more and the whole cycle will have to be repeated. However, this is part of life.

22.2 Using DCG notation, write a grammar for a language that is just like $\mathcal{E}_1$, except that it enforces agreement between the subject and verb of a sentence and thus does not generate “I smells the wumpus.”

22.3 Augment the $\mathcal{E}_1$ grammar so that it handles article–noun agreement. That is, make sure that “agents” is an $NP$, but “agent” and “an agents” are not.

22.4 Outline the major differences between Java (or any other computer language with which you are familiar) and English, commenting on the “understanding” problem in each case. Think about such things as grammar, syntax, semantics, pragmatics, compositionality, context-dependence, lexical ambiguity, syntactic ambiguity, reference finding (including pronouns), background knowledge, and what it means to “understand” in the first place.

22.5 Which of the following are reasons for introducing a quasi-logical form?
   a. To make it easier to write simple compositional grammar rules.
   b. To extend the expressiveness of the semantic representation language.
   c. To be able to represent quantifier scoping ambiguities (among others) in a succinct form.
   d. To make it easier to do semantic disambiguation.

22.6 Determine what semantic interpretation would be given to the following sentences by the grammar in this chapter:
   a. It is a wumpus.
   b. The wumpus is dead.
   c. The wumpus is in 2,2.

Would it be a good idea to have the semantic interpretation for “It is a wumpus” be simply $\exists x: x \in Wumpuses$? Consider alternative sentences such as “It was a wumpus.”
22.7 Without looking back at Exercise 22.1, answer the following questions:
   a. What are the four steps that are mentioned?
   b. What step is left out?
   c. What is “the material” that is mentioned in the text?
   d. What kind of mistake would be expensive?
   e. Is it better to do too few or too many? Why?

22.8 This exercise concerns grammars for very simple languages.
   a. Write a context-free grammar for the language \(a^n b^n\).
   b. Write a context-free grammar for the palindrome language: the set of all strings whose
      second half is the reverse of the first half.
   c. Write a context-sensitive grammar for the duplicate language: the set of all strings
      whose second half is the same as the first half.

22.9 Consider the sentence “Someone walked slowly to the supermarket” and the following
lexicon:

\[
\begin{align*}
\text{Pronoun} & \rightarrow \text{someone} \\
\text{Adv} & \rightarrow \text{slowly} \\
\text{Det} & \rightarrow \text{the} \\
\text{V} & \rightarrow \text{walked} \\
\text{Prep} & \rightarrow \text{to} \\
\text{Noun} & \rightarrow \text{supermarket}
\end{align*}
\]

Which of the following three grammars, combined with the lexicon, generates the given
sentence? Show the corresponding parse tree(s).

(A):
\[
\begin{align*}
S & \rightarrow NP \ VP \\
NP & \rightarrow \text{Pronoun} \\
NP & \rightarrow \text{Article} \ Noun \\
VP & \rightarrow \text{VP} \ PP \\
VP & \rightarrow \text{VP} \ Adv \ Adv \\
VP & \rightarrow \text{Verb} \\
PP & \rightarrow \text{Prep} \ NP \\
NP & \rightarrow \text{Noun}
\end{align*}
\]

(B):
\[
\begin{align*}
S & \rightarrow NP \ VP \\
NP & \rightarrow \text{Pronoun} \\
NP & \rightarrow \text{Article} \ Noun \\
VP & \rightarrow \text{Verb} \ Vmod \\
VP & \rightarrow \text{Adv} \ Vmod \\
VP & \rightarrow \text{Prep} \ NP \\
NP & \rightarrow \text{Noun}
\end{align*}
\]

(C):
\[
\begin{align*}
S & \rightarrow NP \ VP \\
NP & \rightarrow \text{Pronoun} \\
NP & \rightarrow \text{Article} \ NP \\
VP & \rightarrow \text{Verb} \ Adv \\
Adv & \rightarrow \text{PP} \\
PP & \rightarrow \text{Prep} \ NP \\
NP & \rightarrow \text{Noun}
\end{align*}
\]

For each of the preceding three grammars, write down three sentences of English and three
sentences of non-English generated by the grammar. Each sentence should be significantly
different, should be at least six words long, and should be based on an entirely new set of
lexical entries (which you should define). Suggest ways to improve each grammar to avoid
generating the non-English sentences.

22.10 Implement a version of the chart-parsing algorithm that returns a packed tree of all
edges that span the entire input.

22.11 Implement a version of the chart-parsing algorithm that returns a packed tree for the
longest leftmost edge, and then if that edge does not span the whole input, continues the parse
from the end of that edge. Show why you will need to call PREDICT before continuing. The final result is a list of packed trees such that the list as a whole spans the input.

22.12 (Derived from Barton et al. (1987).) This exercise concerns a language we call Buffalo\(^n\), which is very much like English (or at least E\(_0\)), except that the only word in its lexicon is buffalo. Here are two sentences from the language:

- Buffalo buffalo buffalo Buffalo buffalo.
- Buffalo Buffalo buffalo buffalo Buffalo buffalo.

In case you don’t believe these are sentences, here are two English sentences with corresponding syntactic structure:

- Dallas cattle bewilder Denver cattle.
- Chefs London critics admire cook French food.

Write a grammar for Buffalo\(^n\). The lexical categories are city, plural noun, and (transitive) verb, and there should be one grammar rule for sentence, one for verb phrase, and three for noun phrase: plural noun, noun phrase preceded by a city as a modifier, and noun phrase followed by a reduced relative clause. A reduced relative clause is a clause that is missing the relative pronoun. In addition, the clause consists of a subject noun phrase followed by a verb without an object. An example reduced relative clause is “London critics admire” in the example above. Tabulate the number of possible parses for Buffalo\(^n\) for \(n\) up to 10. Extra credit: Carl de Marcken calculated that there are 121,030,872,213,055,159,681,184,485 Buffalo\(^n\) sentences of length 200 (for the grammar he used). How did he do that?

22.13 Draw a discourse parse tree for the story on page 823 about John going to a fancy restaurant. Use to the two grammar rules for Segment, giving the proper CoherenceRelation for each node. (You needn’t show the parse for individual sentences.) Now do the same for a 5 to 10-sentence discourse of your choosing.

22.14 We forgot to mention that the text in Exercise 22.1 is entitled “Washing Clothes.” Reread the text and answer the questions in Exercise 22.7. Did you do better this time? Bransford and Johnson (1973) used this text in a better controlled experiment and found that the title helped significantly. What does this tell you about discourse comprehension?