23 NATURAL LANGUAGE FOR COMMUNICATION

In which we see how humans communicate with one another in natural language, and how computer agents might join in the conversation.

COMMUNICATION
SIGN

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. Humans are the most chatty of all species, and if computer agents are to be helpful, they'll need to learn to speak the language. In this chapter we look at language models for communication. Models aimed at deep understanding of a conversation necessarily need to be more complex than the simple models aimed at, say, spam classification. We start with grammatical models of the phrase structure of sentences, add semantics to the model, and then apply it to machine translation and speech recognition.

23.1 PHRASE STRUCTURE GRAMMARS

The n-gram language models of Chapter 22 were based on sequences of words. The big issue for these models is **data sparsity**—with a vocabulary of, say, 10^5 words, there are 10^{15} trigram probabilities to estimate, and so a corpus of even a trillion words will not be able to supply reliable estimates for all of them. We can address the problem of sparsity through generalization. From the fact that "black dog" is more frequent than "dog black" and similar observations, we can form the generalization that adjectives tend to come before nouns in English (whereas they tend to follow nouns in French: "chien noir" is more frequent). Of course there are always exceptions; "galore" is an adjective that follows the noun it modifies.

LEXICAL CATEGORY
SYNTACTIC CATEGORIES
PHRASE STRUCTURE

Despite the exceptions, the notion of a **lexical category** (also known as a **part of speech**) such as *noun* or *adjective* is a useful generalization—useful in its own right, but more so when we string together lexical categories to form **syntactic categories** such as *noun phrase* or *verb phrase*, and combine these syntactic categories into trees representing the **phrase structure** of sentences: nested phrases, each marked with a category.

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:

Recursively enumerable grammars use unrestricted rules: both sides of the rewrite rules can have any number of terminal and nonterminal symbols, as in the rule $A\ B\ C \rightarrow D\ E$. 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 $A\ X\ B \rightarrow A\ Y\ B$ says that an X can be rewritten as a Y 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 bs and then cs).

In **context-free grammars** (or **CFG**s), 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 as followed by any number of bs.

The grammars higher up in the hierarchy have more expressive power, but the algorithms for dealing with them are less efficient. Up to the 1980s, linguists focused on context-free and context-sensitive languages. Since then, there has been renewed interest in regular grammars, brought about by the need to process and learn from gigabytes or terabytes 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) (and note that these three authors all now work on large text corpora at Google).

PROBABILISTIC CONTEXT-FREE GRAMMAR

GRAMMAR

LANGUAGE

There have been many competing language models based on the idea of phrase structure; we will describe a popular model called the **probabilistic context-free grammar**, or PCFG.[1] A **grammar** is a collection of rules that defines a **language** as a set of allowable strings of words. "Context-free" is described in the sidebar on page 889, and "probabilistic" means that the grammar assigns a probability to every string. Here is a PCFG rule:

$$\begin{aligned} VP \;\rightarrow\;& \mathit{Verb}\ [0.70] \\ \mid\;& VP\ NP\ [0.30]\ . \end{aligned}$$

NON-TERMINAL SYMBOLS

TERMINAL SYMBOL

Here VP (*verb phrase*) and NP (*noun phrase*) are **non-terminal symbols**. The grammar also refers to actual words, which are called **terminal symbols**. This rule is saying that with probability 0.70 a verb phrase consists solely of a verb, and with probability 0.30 it is a VP followed by an NP. Appendix B describes non-probabilistic context-free grammars.

We now define a grammar for a tiny fragment of English that is suitable for communication between agents exploring the wumpus world. We call this language $\mathcal{E}_0$. Later sections improve on $\mathcal{E}_0$ to make it slightly 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.

23.1.1 The lexicon of $\mathcal{E}_0$

LEXICON

First we define the **lexicon**, or list of allowable words. The words are grouped into the lexical categories familiar to dictionary users: nouns, pronouns, and names to denote things; verbs to denote events; adjectives to modify nouns; adverbs to modify verbs; and function words: articles (such as *the*), prepositions (*in*), and conjunctions (*and*). Figure 23.1 shows a small lexicon for the language $\mathcal{E}_0$.

OPEN CLASS

CLOSED CLASS

Each of the categories ends in ... to indicate that there are other words in the category. For nouns, names, verbs, adjectives, and adverbs, it is infeasible even in principle to list all the words. Not only are there tens of thousands of members in each class, but new ones–like *iPod* or *biodiesel*—are being added constantly. These five categories are called **open classes**. For the categories of pronoun, relative pronoun, article, preposition, and conjunction we could have listed all the words with a little more work. These are called **closed classes**; they have a small number of words (a dozen or so). 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.

23.1.2 The Grammar of $\mathcal{E}_0$

PARSE TREE

The next step is to combine the words into phrases. Figure 23.2 shows a grammar for $\mathcal{E}_0$, with rules for each of the six syntactic categories and an example for each rewrite rule.[2] Figure 23.3 shows a **parse tree** for the sentence "Every wumpus smells." The parse tree

[1] PCFGs are also known as stochastic context-free grammars, or SCFGs.

[2] 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. An example of a relative clause is *that stinks* in "The wumpus *that stinks* is in 2 2." Another kind of relative clause has no relative pronoun, e.g., *I know* in "the man *I know*."

Noun	$\rightarrow$	**stench** [0.05] | **breeze** [0.10] | **wumpus** [0.15] | **pits** [0.05] | ...
Verb	$\rightarrow$	**is** [0.10] | **feel** [0.10] | **smells** [0.10] | **stinks** [0.05] | ...
Adjective	$\rightarrow$	**right** [0.10] | **dead** [0.05] | **smelly** [0.02] | **breezy** [0.02] ...
Adverb	$\rightarrow$	**here** [0.05] | **ahead** [0.05] | **nearby** [0.02] | ...
Pronoun	$\rightarrow$	**me** [0.10] | **you** [0.03] | **I** [0.10] | **it** [0.10] | ...
RelPro	$\rightarrow$	**that** [0.40] | **which** [0.15] | **who** [0.20] | **whom** [0.02] $\vee$ ...
Name	$\rightarrow$	**John** [0.01] | **Mary** [0.01] | **Boston** [0.01] | ...
Article	$\rightarrow$	**the** [0.40] | **a** [0.30] | **an** [0.10] | **every** [0.05] | ...
Prep	$\rightarrow$	**to** [0.20] | **in** [0.10] | **on** [0.05] | **near** [0.10] | ...
Conj	$\rightarrow$	**and** [0.50] | **or** [0.10] | **but** [0.20] | **yet** [0.02] $\vee$ ...
Digit	$\rightarrow$	**0** [0.20] | **1** [0.20] | **2** [0.20] | **3** [0.20] | **4** [0.20] | ...

Figure 23.1 The lexicon for $\mathcal{E}_0$. *RelPro* is short for relative pronoun, *Prep* for preposition, and *Conj* for conjunction. The sum of the probabilities for each category is 1.

$\mathcal{E}_0$:	*S*	$\rightarrow$	*NP VP*	[0.90]	I + feel a breeze
		|	*S Conj S*	[0.10]	I feel a breeze + and + It stinks
	NP	$\rightarrow$	*Pronoun*	[0.30]	I
		|	*Name*	[0.10]	John
		|	*Noun*	[0.10]	pits
		|	*Article Noun*	[0.25]	the + wumpus
		|	*Article Adjs Noun*	[0.05]	the + smelly dead + wumpus
		|	*Digit Digit*	[0.05]	3 4
		|	*NP PP*	[0.10]	the wumpus + in 1 3
		|	*NP RelClause*	[0.05]	the wumpus + that is smelly
	VP	$\rightarrow$	*Verb*	[0.40]	stinks
		|	*VP NP*	[0.35]	feel + a breeze
		|	*VP Adjective*	[0.05]	smells + dead
		|	*VP PP*	[0.10]	is + in 1 3
		|	*VP Adverb*	[0.10]	go + ahead
	Adjs	$\rightarrow$	*Adjective*	[0.80]	smelly
		|	*Adjective Adjs*	[0.20]	smelly + dead
	PP	$\rightarrow$	*Prep NP*	[1.00]	to + the east
	RelClause	$\rightarrow$	*RelPro VP*	[1.00]	that + is smelly

Figure 23.2 The grammar for $\mathcal{E}_0$, with example phrases for each rule. The syntactic categories are sentence (*S*), noun phrase (*NP*), verb phrase (*VP*), list of adjectives (*Adjs*), prepositional phrase (*PP*), and relative clause (*RelClause*).

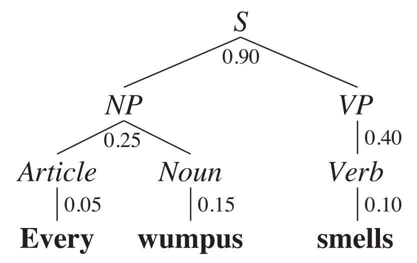


Figure 23.3 Parse tree for the sentence "Every wumpus smells" according to the grammar $\mathcal{E}_0$. Each interior node of the tree is labeled with its probability. The probability of the tree as a whole is $0.9 \times 0.25 \times 0.05 \times 0.15 \times 0.40 \times 0.10 = 0.0000675$. Since this tree is the only parse of the sentence, that number is also the probability of the sentence. The tree can also be written in linear form as $[S\ [NP\ [Article$ **every**$]\ [Noun$ **wumpus**$]][VP\ [Verb$ **smells**$]]]$.

gives a constructive proof that the string of words is indeed a sentence according to the rules of $\mathcal{E}_0$. The $\mathcal{E}_0$ grammar generates a wide range of English sentences such as the following:

John is in the pit
The wumpus that stinks is in 2 2
Mary is in Boston and the wumpus is near 3 2

OVERGENERATION
UNDERGENERATION

Unfortunately, the grammar **overgenerates**: that is, it generates sentences that are not grammatical, such as "Me go Boston" and "I smell pits wumpus John." It also **undergenerates**: there are many sentences of English that it rejects, such as "I think the wumpus is smelly." We will see how to learn a better grammar later; for now we concentrate on what we can do with the grammar we have.

23.2 Syntactic Analysis (Parsing)

PARSING

Parsing is the process of analyzing a string of words to uncover its phrase structure, according to the rules of a grammar. Figure 23.4 shows that we can start with the S symbol and search top down for a tree that has the words as its leaves, or we can start with the words and search bottom up for a tree that culminates in an S. Both top-down and bottom-up parsing can be inefficient, however, because they can end up repeating effort in areas of the search space that lead to dead ends. 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 and reanalyze the whole sentence under the other interpretation.

List of items	*Rule*
S	
$NP\ VP$	$S \rightarrow NP\ VP$
$NP\ VP\ Adjective$	$VP \rightarrow VP\ Adjective$
$NP\ Verb\ Adjective$	$VP \rightarrow Verb$
$NP\ Verb$ **dead**	$Adjective \rightarrow$ **dead**
NP **is dead**	$Verb \rightarrow$ **is**
$Article\ Noun$ **is dead**	$NP \rightarrow Article\ Noun$
$Article$ **wumpus is dead**	$Noun \rightarrow$ **wumpus**
the wumpus is dead	$Article \rightarrow$ **the**

Figure 23.4 Trace of the process of finding a parse for the string "The wumpus is dead" as a sentence, according to the grammar $\mathcal{E}_0$. Viewed as a top-down parse, we start with the list of items being S and, on each step, match an item X with a rule of the form $(X \rightarrow \dots)$ and replace X in the list of items with $(\dots)$. Viewed as a bottom-up parse, we start with the list of items being the words of the sentence, and, on each step, match a string of tokens $(\dots)$ in the list against a rule of the form $(X \rightarrow \dots)$ and replace $(\dots)$ with X.

To avoid this source of inefficiency we can use dynamic programming: *every time we analyze a substring, store the results so we won't have to reanalyze 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. There are many types of chart parsers; we describe a bottom-up version called the **CYK algorithm**, after its inventors, John Cocke, Daniel Younger, and Tadeo Kasami.

CHART

CYK ALGORITHM

The CYK algorithm is shown in Figure 23.5. Note that it requires a grammar with all rules in one of two very specific formats: lexical rules of the form $X \rightarrow$ **word**, and syntactic rules of the form $X \rightarrow Y\ Z$. This grammar format, called **Chomsky Normal Form**, may seem restrictive, but it is not: any context-free grammar can be automatically transformed into Chomsky Normal Form. Exercise 23.8 leads you through the process.

CHOMSKY NORMAL FORM

The CYK algorithm uses space of $O(n^2m)$ for the P table, where n is the number of words in the sentence, and m is the number of nonterminal symbols in the grammar, and takes time $O(n^3m)$. (Since m is constant for a particular grammar, this is commonly described as $O(n^3)$.) No algorithm can do better for general context-free grammars, although there are faster algorithms on more restricted grammars. In fact, it is quite a trick for the algorithm to complete in $O(n^3)$ time, given that it is possible for a sentence to have an exponential number of parse trees. Consider the sentence

Fall leaves fall and spring leaves spring.

It is ambiguous because each word (except "and") can be either a noun or a verb, and "fall" and "spring" can be adjectives as well. (For example, one meaning of "Fall leaves fall" is

```
function CYK-PARSE($words$, $grammar$) returns $P$, a table of probabilities
  $N \leftarrow$ LENGTH($words$)
  $M \leftarrow$ the number of nonterminal symbols in $grammar$
  $P \leftarrow$ an array of size $[M, N, N]$, initially all 0
  /* Insert lexical rules for each word */
  for $i$ = 1 to $N$ do
    for each rule of form ($X \rightarrow words_i\ [p]$) do
      $P[X, i, 1] \leftarrow$ p
  /* Combine first and second parts of right-hand sides of rules, from short to long */
  for $length$ = 2 to $N$ do
    for $start$ = 1 to $N - length + 1$ do
      for $len1$ = 1 to $N - 1$ do
        $len2 \leftarrow length - len1$
        for each rule of the form ($X \rightarrow Y\ Z\ [p]$) do
          $P[X, start, length] \leftarrow$ MAX($P[X, start, length]$,
                              $P[Y, start, len1] \times P[Z, start + len1, len2] \times p$)
  return $P$
```

Figure 23.5 The CYK algorithm for parsing. Given a sequence of words, it finds the most probable derivation for the whole sequence and for each subsequence. It returns the whole table, P, in which an entry $P[X, start, len]$ is the probability of the most probable X of length len starting at position $start$. If there is no X of that size at that location, the probability is 0.

equivalent to “Autumn abandons autumn.) With $\mathcal{E}_0$ 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 c two-ways-ambiguous conjoined subsentences, we would have 2^c ways of choosing parses for the subsentences.[3] How does the CYK algorithm process these 2^c parse trees in $O(c^3)$ time? The answer is that it doesn't examine all the parse trees; all it has to do is compute the probability of the most probable tree. The subtrees are all represented in the P table, and with a little work we could enumerate them all (in exponential time), but the beauty of the CYK algorithm is that we don't have to enumerate them unless we want to.

In practice we are usually not interested in all parses; just the best one or best few. Think of the CYK algorithm as defining the complete state space defined by the “apply grammar rule” operator. It is possible to search just part of this space using A^* search. Each state in this space is a list of items (words or categories), as shown in the bottom-up parse table (Figure 23.4). The start state is a list of words, and a goal state is the single item S. The

[3] There also would be $O(c!)$ ambiguity in the way the components conjoin—for example, (X and (Y and Z)) versus ((X and Y) and Z). But that is another story, one told well by Church and Patil (1982).

```
[ [S [NP-SBJ-2 Her eyes]
    [VP were
       [VP glazed
          [NP *-2]
          [SBAR-ADV as if
              [S [NP-SBJ she]
                 [VP did n't
                    [VP [VP hear [NP *-1]]
                        or
                        [VP [ADVP even] see [NP *-1]]
                        [NP-1 him]]]]]]]]
  .]
```

Figure 23.6 Annotated tree for the sentence "Her eyes were glazed as if she didn't hear or even see him." from the Penn Treebank. Note that in this grammar there is a distinction between an object noun phrase (*NP*) and a subject noun phrase (*NP-SBJ*). Note also a grammatical phenomenon we have not covered yet: the movement of a phrase from one part of the tree to another. This tree analyzes the phrase "hear or even see him" as consisting of two constituent VPs, [VP hear [NP *-1]] and [VP [ADVP even] see [NP *-1]], both of which have a missing object, denoted *-1, which refers to the NP labeled elsewhere in the tree as [NP-1 him].

cost of a state is the inverse of its probability as defined by the rules applied so far, and there are various heuristics to estimate the remaining distance to the goal; the best heuristics come from machine learning applied to a corpus of sentences. With the A^* algorithm we don't have to search the entire state space, and we are guaranteed that the first parse found will be the most probable.

23.2.1 Learning probabilities for PCFGs

A PCFG has many rules, with a probability for each rule. This suggests that **learning** the grammar from data might be better than a knowledge engineering approach. Learning is easiest if we are given a corpus of correctly parsed sentences, commonly called a **treebank**. The Penn Treebank (Marcus *et al.*, 1993) is the best known; it consists of 3 million words which have been annotated with part of speech and parse-tree structure, using human labor assisted by some automated tools. Figure 23.6 shows an annotated tree from the Penn Treebank.

TREEBANK

Given a corpus of trees, we can create a PCFG just by counting (and smoothing). In the example above, there are two nodes of the form $[S[NP\ldots][VP\ldots]]$. We would count these, and all the other subtrees with root S in the corpus. If there are 100,000 S nodes of which 60,000 are of this form, then we create the rule:

$$S \rightarrow NP\ VP\ [0.60]\ .$$

What if a treebank is not available, but we have a corpus of raw unlabeled sentences? It is still possible to learn a grammar from such a corpus, but it is more difficult. First of all, we actually have two problems: learning the structure of the grammar rules and learning the probabilities associated with each rule. (We have the same distinction in learning Bayes nets.) We'll assume that we're given the lexical and syntactic category names. (If not, we can just assume categories $X_1, \ldots X_n$ and use cross-validation to pick the best value of n.) We can then assume that the grammar includes every possible ($X \;\rightarrow\; Y\ Z$) or ($X \;\rightarrow$ *word*) rule, although many of these rules will have probability 0 or close to 0.

We can then use an expectation–maximization (EM) approach, just as we did in learning HMMs. The parameters we are trying to learn are the rule probabilities; we start them off at random or uniform values. The hidden variables are the parse trees: we don't know whether a string of words $w_i \ldots w_j$ is or is not generated by a rule ($X \;\rightarrow\; \ldots$). The E step estimates the probability that each subsequence is generated by each rule. The M step then estimates the probability of each rule. The whole computation can be done in a dynamic-programming fashion with an algorithm called the **inside–outside algorithm** in analogy to the forward–backward algorithm for HMMs.

INSIDE–OUTSIDE ALGORITHM

The inside–outside algorithm seems magical in that it induces a grammar from unparsed text. But it has several drawbacks. First, the parses that are assigned by the induced grammars are often difficult to understand and unsatisfying to linguists. This makes it hard to combine handcrafted knowledge with automated induction. Second, it is slow: $O(n^3 m^3)$, where n is the number of words in a sentence and m is the number of grammar categories. Third, the space of probability assignments is very large, and empirically it seems that getting stuck in local maxima is a severe problem. Alternatives such as simulated annealing can get closer to the global maximum, at a cost of even more computation. Lari and Young (1990) conclude that inside–outside is "computationally intractable for realistic problems."

However, progress can be made if we are willing to step outside the bounds of learning solely from unparsed text. One approach is to learn from **prototypes**: to seed the process with a dozen or two rules, similar to the rules in $\mathcal{E}_1$. From there, more complex rules can be learned more easily, and the resulting grammar parses English with an overall recall and precision for sentences of about 80% (Haghighi and Klein, 2006). Another approach is to use treebanks, but in addition to learning PCFG rules directly from the bracketings, also learning distinctions that are not in the treebank. For example, not that the tree in Figure 23.6 makes the distinction between NP and $NP - SBJ$. The latter is used for the pronoun "she," the former for the pronoun "her." We will explore this issue in Section 23.6; for now let us just say that there are many ways in which it would be useful to **split** a category like NP—grammar induction systems that use treebanks but automatically split categories do better than those that stick with the original category set (Petrov and Klein, 2007c). The error rates for automatically learned grammars are still about 50% higher than for hand-constructed grammar, but the gap is decreasing.

23.2.2 Comparing context-free and Markov models

The problem with PCFGs is that they are context-free. That means that the difference between P("eat a banana") and P("eat a bandanna") depends only on $P(Noun \;\rightarrow$ "banana") versus $P(Noun \;\rightarrow$ "bandanna") and not on the relation between "eat" and the respective objects. A Markov model of order two or more, given a sufficiently large corpus, *will* know that "eat a banana" is more probable. We can combine a PCFG and Markov model to get the best of both. The simplest approach is to estimate the probability of a sentence with the geometric mean of the probabilities computed by both models. Then we would know that "eat a banana" is probable from both the grammatical and lexical point of view. But it still wouldn't pick up the relation between "eat" and "banana" in "eat a slightly aging but still palatable banana" because here the relation is more than two words away. Increasing the order of the Markov model won't get at the relation precisely; to do that we can use a **lexicalized** PCFG, as described in the next section.

Another problem with PCFGs is that they tend to have too strong a preference for shorter sentences. In a corpus such as the *Wall Street Journal*, the average length of a sentence is about 25 words. But a PCFG will usually assign fairly high probability to many short sentences, such as "He slept," whereas in the *Journal* we're more likely to see something like "It has been reported by a reliable source that the allegation that he slept is credible." It seems that the phrases in the *Journal* really are not context-free; instead the writers have an idea of the expected sentence length and use that length as a soft global constraint on their sentences. This is hard to reflect in a PCFG.

23.3 AUGMENTED GRAMMARS AND SEMANTIC INTERPRETATION

In this section we see how to extend context-free grammars—to say that, for example, not every NP is independent of context, but rather, certain NPs are more likely to appear in one context, and others in another context.

23.3.1 Lexicalized PCFGs

LEXICALIZED PCFG

HEAD

AUGMENTED GRAMMAR

To get at the relationship between the verb "eat" and the nouns "banana" versus "bandanna," we can use a **lexicalized PCFG**, in which the probabilities for a rule depend on the relationship between words in the parse tree, not just on the adjacency of words in a sentence. Of course, we can't have the probability depend on every word in the tree, because we won't have enough training data to estimate all those probabilities. It is useful to introduce the notion of the **head** of a phrase—the most important word. Thus, "eat" is the head of the VP "eat a banana" and "banana" is the head of the NP "a banana." We use the notation $VP(v)$ to denote a phrase with category VP whose head word is v. We say that the category VP is **augmented** with the head variable v. Here is an **augmented grammar** that describes the verb–object relation:

$$\begin{array}{lcll}
VP(v) & \rightarrow & Verb(v)\ NP(n) & [P_1(v,n)] \\
VP(v) & \rightarrow & Verb(v) & [P_2(v)] \\
NP(n) & \rightarrow & Article(a)\ Adjs(j)\ Noun(n) & [P_3(n,a)] \\
Noun(\textbf{banana}) & \rightarrow & \textbf{banana} & [p_n] \\
\ldots & & & \ldots
\end{array}$$

Here the probability $P_1(v, n)$ depends on the head words v and n. We would set this probability to be relatively high when v is "eat" and n is "banana," and low when n is "bandanna."

Note that since we are considering only heads, the distinction between "eat a banana" and "eat a rancid banana" will not be caught by these probabilities. Another issue with this approach is that, in a vocabulary with, say, 20,000 nouns and 5,000 verbs, P_1 needs 100 million probability estimates. Only a few percent of these can come from a corpus; the rest will have to come from smoothing (see Section 22.1.2). For example, we can estimate$P_1(v, n)$ for a (v, n) pair that we have not seen often (or at all) by backing off to a model that depends only on v. These objectless probabilities are still very useful; they can capture the distinction between a transitive verb like "eat"—which will have a high value for P_1 and a low value for P_2—and an intransitive verb like "sleep," which will have the reverse. It is quite feasible to learn these probabilities from a treebank.

23.3.2 Formal definition of augmented grammar rules

Augmented rules are complicated, so we will give them a formal definition by showing how an augmented rule can be translated into a logical sentence. The sentence will have the form of a definite clause (see page 256), so the result is called a **definite clause grammar**, or DCG. We'll use as an example a version of a rule from the lexicalized grammar for NP with one new piece of notation:

DEFINITE CLAUSE GRAMMAR

$$NP(n) \;\rightarrow\; Article(a)\; Adjs(j)\; Noun(n)\; \{Compatible(j, n)\}\ .$$

The new aspect here is the notation $\{constraint\}$ to denote a logical constraint on some of the variables; the rule only holds when the constraint is true. Here the predicate $Compatible(j, n)$ is meant to test whether adjective j and noun n are compatible; it would be defined by a series of assertions such as $Compatible(\mathbf{black}, \mathbf{dog})$. We can convert this grammar rule into a definite clause by (1) reversing the order of right- and left-hand sides, (2) making a conjunction of all the constituents and constraints, (3) adding a variable s_i to the list of arguments for each constituent to represent the sequence of words spanned by the constituent, (4) adding a term for the concatenation of words, $Append(s_1, \ldots)$, to the list of arguments for the root of the tree. That gives us

$$\begin{aligned} &Article(a, s_1) \wedge Adjs(j, s_2) \wedge Noun(n, s_3) \wedge Compatible(j, n) \\ &\qquad \Rightarrow\; NP(n, Append(s_1, s_2, s_3))\ . \end{aligned}$$

This definite clause says that if the predicate $Article$ is true of a head word a and a string s_1, and $Adjs$ is similarly true of a head word j and a string s_2, and $Noun$ is true of a head word n and a string s_3, and if j and n are compatible, then the predicate NP is true of the head word n and the result of appending strings s_1, s_2, and s_3.

The DCG translation left out the probabilities, but we could put them back in: just augment each constituent with one more variable representing the probability of the constituent, and augment the root with a variable that is the product of the constituent probabilities times the rule probability.

The translation from grammar rule to definite clause allows us to talk about parsing as logical inference. This makes it possible to reason about languages and strings in many different ways. For example, it means we can do bottom-up parsing using forward chaining or top-down parsing using backward chaining. In fact, parsing natural language with DCGs was

LANGUAGE GENERATION

one of the first applications of (and motivations for) the Prolog logic programming language. It is sometimes possible to run the process backward and do **language generation** as well as parsing. For example, skipping ahead to Figure 23.10 (page 903), a logic program could be given the semantic form $Loves(John, Mary)$ and apply the definite-clause rules to deduce

$$S(Loves(John, Mary), [\textbf{John}, \textbf{loves}, \textbf{Mary}]) \; .$$

This works for toy examples, but serious language-generation systems need more control over the process than is afforded by the DCG rules alone.

$$
\begin{array}{llcl}
\mathcal{E}_1 : & S & \rightarrow & NP_S \; VP \;|\; \ldots \\
& NP_S & \rightarrow & Pronoun_S \;|\; Name \;|\; Noun \;|\; \ldots \\
& NP_O & \rightarrow & Pronoun_O \;|\; Name \;|\; Noun \;|\; \ldots \\
& VP & \rightarrow & VP \; NP_O \;|\; \ldots \\
& PP & \rightarrow & Prep \; NP_O \\
& Pronoun_S & \rightarrow & \textbf{I} \;|\; \textbf{you} \;|\; \textbf{he} \;|\; \textbf{she} \;|\; \textbf{it} \;|\; \ldots \\
& Pronoun_O & \rightarrow & \textbf{me} \;|\; \textbf{you} \;|\; \textbf{him} \;|\; \textbf{her} \;|\; \textbf{it} \;|\; \ldots \\
& & & \ldots
\end{array}
$$

$$
\begin{array}{llcl}
\mathcal{E}_2 : & S(head) & \rightarrow & NP(Sbj, pn, h) \; VP(pn, head) \;|\; \ldots \\
& NP(c, pn, head) & \rightarrow & Pronoun(c, pn, head) \;|\; Noun(c, pn, head) \;|\; \ldots \\
& VP(pn, head) & \rightarrow & VP(pn, head) \; NP(Obj, p, h) \;|\; \ldots \\
& PP(head) & \rightarrow & Prep(head) \; NP(Obj, pn, h) \\
& Pronoun(Sbj, 1S, \textbf{I}) & \rightarrow & \textbf{I} \\
& Pronoun(Sbj, 1P, \textbf{we}) & \rightarrow & \textbf{we} \\
& Pronoun(Obj, 1S, \textbf{me}) & \rightarrow & \textbf{me} \\
& Pronoun(Obj, 3P, \textbf{them}) & \rightarrow & \textbf{them} \\
& & & \ldots
\end{array}
$$

Figure 23.7 Top: part of a grammar for the language $\mathcal{E}_1$, which handles subjective and objective cases in noun phrases and thus does not overgenerate quite as badly as $\mathcal{E}_0$. The portions that are identical to $\mathcal{E}_0$ have been omitted. Bottom: part of an augmented grammar for $\mathcal{E}_2$, with three augmentations: case agreement, subject–verb agreement, and head word. *Sbj, Obj, 1S, 1P* and *3P* are constants, and lowercase names are variables.

23.3.3 Case agreement and subject–verb agreement

We saw in Section 23.1 that the simple grammar for $\mathcal{E}_0$ overgenerates, producing nonsentences 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.[4] We can account for this by

[4] 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.

splitting NP into two categories, 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 23.7 shows the 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.

CASE AGREEMENT

SUBJECT–VERB AGREEMENT

Unfortunately, $\mathcal{E}_1$ still overgenerates. English requires **subject–verb agreement** for person and number of 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: the verb "to be" has three forms, "I am / you are / he is." So one distinction (case) splits NP two ways, another distinction (person and number) splits NP three ways, and as we uncover other distinctions we would end up with an exponential number of subscripted NP forms if we took the approach of $\mathcal{E}_1$. Augmentations are a better approach: they can represent an exponential number of forms as a single rule.

In the bottom of Figure 23.7 we see (part of) an augmented grammar for the language $\mathcal{E}_2$, which handles case agreement, subject–verb agreement, and head words. We have just one NP category, but $NP(c, pn, head)$ has three augmentations: c is a parameter for case, pn is a parameter for person and number, and $head$ is a parameter for the head word of the phrase. The other categories also are augmented with heads and other arguments. Let's consider one rule in detail:

$$S(head) \rightarrow NP(Sbj, pn, h)\ VP(pn, head)\ .$$

This rule is easiest to understand right-to-left: when an *NP* and a *VP* are conjoined they form an *S*, but only if the *NP* has the subjective (*Sbj*) case and the person and number (*pn*) of the *NP* and *VP* are identical. If that holds, then we have an *S* whose head is the same as the head of the *VP*. Note the head of the *NP*, denoted by the dummy variable *h*, is not part of the augmentation of the *S*. The lexical rules for $\mathcal{E}_2$ fill in the values of the parameters and are also best read right-to-left. For example, the rule

$$Pronoun(Sbj, 1S, \mathbf{I}) \rightarrow \mathbf{I}$$

says that "I" can be interpreted as a *Pronoun* in the subjective case, first-person singular, with head "I." For simplicity we have omitted the probabilities for these rules, but augmentation does work with probabilities. Augmentation can also work with automated learning mechanisms. Petrov and Klein (2007c) show how a learning algorithm can automatically split the NP category into NP_S and NP_O.

23.3.4 Semantic interpretation

To show how to add semantics to a grammar, we start with an example that is simpler than English: the semantics of arithmetic expressions. Figure 23.8 shows a grammar for arithmetic expressions, where each rule is augmented with a variable indicating the semantic interpretation of the phrase. The semantics of a digit such as "3" is the digit itself. The semantics of an expression such as "3 + 4" is the operator "+" applied to the semantics of the phrase "3" and

$$
\begin{aligned}
&\mathit{Exp}(x) \;\rightarrow\; \mathit{Exp}(x_1)\ \mathit{Operator}(op)\ \mathit{Exp}(x_2)\ \{x = \mathit{Apply}(op, x_1, x_2)\}\\
&\mathit{Exp}(x) \;\rightarrow\; \textbf{(}\ \mathit{Exp}(x)\ \textbf{)}\\
&\mathit{Exp}(x) \;\rightarrow\; \mathit{Number}(x)\\
&\mathit{Number}(x) \;\rightarrow\; \mathit{Digit}(x)\\
&\mathit{Number}(x) \;\rightarrow\; \mathit{Number}(x_1)\ \mathit{Digit}(x_2)\ \{x = 10 \times x_1 + x_2\}\\
&\mathit{Digit}(x) \;\rightarrow\; x\ \{0 \le x \le 9\}\\
&\mathit{Operator}(x) \;\rightarrow\; x\ \{x \in \{+, -, \div, \times\}\}
\end{aligned}
$$

Figure 23.8 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.

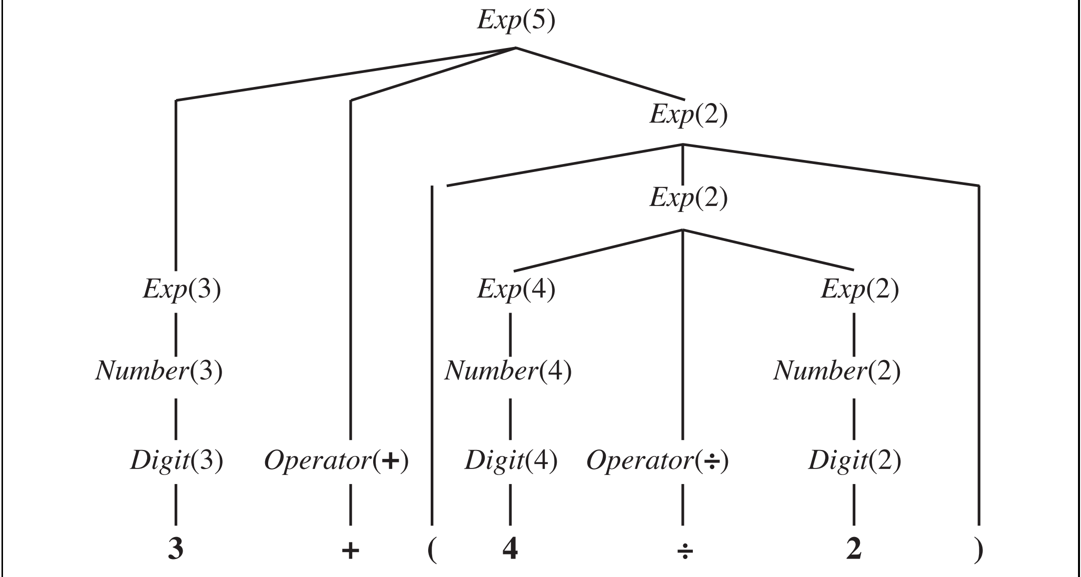


Figure 23.9 Parse tree with semantic interpretations for the string "$3 + (4 \div 2)$".

COMPOSITIONAL SEMANTICS

the phrase "4." The rules obey the principle of **compositional semantics**—the semantics of a phrase is a function of the semantics of the subphrases. Figure 23.9 shows the parse tree for $3 + (4 \div 2)$ according to this grammar. The root of the parse tree is $\mathit{Exp}(5)$, an expression whose semantic interpretation is 5.

Now let's move on to the semantics of English, or at least of $\mathcal{E}_0$. We start by determining what semantic representations we want to associate with what phrases. We use the simple example sentence "John loves Mary." The *NP* "John" should have as its semantic interpretation the logical term John, and the sentence as a whole should have as its interpretation the logical sentence $\mathit{Loves}(\mathit{John}, \mathit{Mary})$. 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 λ-notation (see page 294), we can represent "loves Mary" as the predicate

$$\lambda x \; Loves(x, Mary) \; .$$

Now we need a rule that says "an NP with semantics obj followed by a VP with semantics $pred$ yields a sentence whose semantics is the result of applying $pred$ to obj:"

$$S(pred(obj)) \;\rightarrow\; NP(obj) \;\; VP(pred) \; .$$

The rule tells us that the semantic interpretation of "John loves Mary" is

$$(\lambda x \; Loves(x, Mary))(John) \; ,$$

which is equivalent to $Loves(John, Mary)$.

The rest of the semantics follows in a straightforward way from the choices we have made so far. Because 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 $\lambda y \; \lambda x \; Loves(x, y)$, the predicate that, when given the argument $Mary$, returns the predicate $\lambda x \; Loves(x, Mary)$. We end up with the grammar shown in Figure 23.10 and the parse tree shown in Figure 23.11. We could just as easily have added semantics to $\mathcal{E}_2$; we chose to work with $\mathcal{E}_0$ so that the reader can focus on one type of augmentation at a time.

Adding semantic augmentations to a grammar by hand is laborious and error prone. Therefore, there have been several projects to learn semantic augmentations from examples. CHILL (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 semantic forms—for example;

What is the capital of the state with the largest population?
$Answer(c, Capital(s, c) \wedge Largest(p, State(s) \wedge Population(s, p)))$

CHILL's task is to learn a predicate $Parse(words, semantics)$ 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 $Parse$ 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 can learn to achieve 70% to 85% accuracy on various database query tasks.

23.3.5 Complications

The grammar of real English is endlessly complex. We will briefly mention some examples.

TIME AND TENSE

Time and tense: 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

$$S(pred(obj)) \rightarrow NP(obj)\ VP(pred)$$
$$VP(pred(obj)) \rightarrow Verb(pred)\ NP(obj)$$
$$NP(obj) \rightarrow Name(obj)$$

$$Name(John) \rightarrow \textbf{John}$$
$$Name(Mary) \rightarrow \textbf{Mary}$$
$$Verb(\lambda y\ \lambda x\ Loves(x,y)) \rightarrow \textbf{loves}$$

Figure 23.10 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.

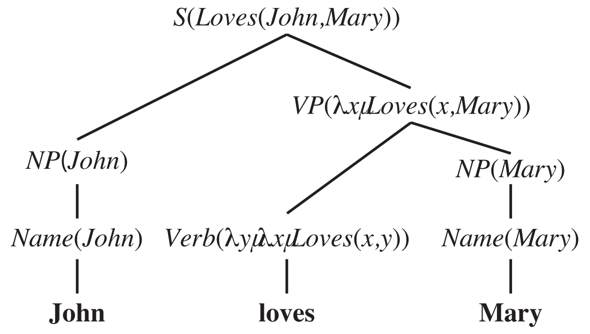


Figure 23.11 A parse tree with semantic interpretations for the string "John loves Mary".

the relative time of an event. One good choice to represent the time of events is the event calculus notation of Section 12.3. In event calculus we have

John loves mary: $E_1 \in Loves(John, Mary) \land During(Now, Extent(E_1))$
John loved mary: $E_2 \in Loves(John, Mary) \land After(Now, Extent(E_2))$.

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

$$Verb(\lambda y\ \lambda x\ e \in Loves(x,y) \land During(Now, e)) \rightarrow \textbf{loves}$$
$$Verb(\lambda y\ \lambda x\ e \in Loves(x,y) \land After(Now, e)) \rightarrow \textbf{loved}\ .$$

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). It is also encouraging that the distinction between processes and discrete events that we made in our discussion of knowledge representation in Section 12.3.1 is actually reflected in language use. We can say "John slept a lot last night," where *Sleeping* is a process category, but it is odd to say "John found a unicorn a lot last night," where *Finding* is a discrete event category. A grammar would reflect that fact by having a low probability for adding the adverbial phrase "a lot" to discrete events.

QUANTIFICATION

Quantification: Consider the sentence "Every agent feels a breeze." The sentence has only one syntactic parse under $\mathcal{E}_0$, but it is actually semantically ambiguous; the preferred meaning is "For every agent there exists a breeze that the agent feels," but an acceptable alternative meaning is "There exists a breeze that every agent feels."[5] The two interpretations can be represented as

$$\begin{array}{l} \forall\, a \;\; a \in Agents \;\Rightarrow \\ \qquad \exists\, b \;\; b \in Breezes \land \exists\, e \;\; e \in Feel(a,b) \land During(Now, e)\;; \\ \exists\, b \;\; b \in Breezes \; \forall\, a \;\; a \in Agents \;\Rightarrow \\ \qquad \exists\, e \;\; e \in Feel(a,b) \land During(Now, e)\;. \end{array}$$

QUASI-LOGICAL FORM

The standard approach to quantification is for the grammar to define not an actual logical semantic sentence, but rather a **quasi-logical form** that is then turned into a logical sentence by algorithms outside of the parsing process. Those algorithms can have preference rules for preferring one quantifier scope over another—preferences that need not be reflected directly in the grammar.

PRAGMATICS

Pragmatics: 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. 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," both "I" and "today" are indexicals. The word "I" would be represented by the fluent $Speaker$, and it would be up to the hearer to resolve the meaning of the fluent—that is not considered part of the grammar but rather an issue of pragmatics; of using the context of the current situation to interpret fluents.

INDEXICAL

SPEECH ACT

Another part of pragmatics is interpreting the speaker's intent. The speaker's action is considered a **speech act**, and it is up to the hearer to decipher what type of action it is—a question, a statement, a promise, a warning, a command, and so on. 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. 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:

$$\begin{array}{l} S(Statement(Speaker, pred(obj))) \;\rightarrow\; NP(obj)\;\; VP(pred) \\ S(Command(Speaker, pred(Hearer))) \;\rightarrow\; VP(pred)\;. \end{array}$$

LONG-DISTANCE DEPENDENCIES

Long-distance dependencies: Questions introduce a new grammatical complexity. In "Who did the agent tell you to give the gold to?" the final word "to" should be parsed as [PP to ␣], where the "␣" denotes a gap or **trace** where an NP is missing; the missing NP is licensed by the first word of the sentence, "who." A complex system of augmentations is used to make sure that the missing NPs match up with the licensing words in just the right way, and prohibit gaps in the wrong places. For example, you can't have a gap in one branch of an NP conjunction: "What did he play [NP Dungeons and ␣]?" is ungrammatical. But you can have the same gap in both branches of a VP conjunction: "What did you [VP [VP smell ␣] and [VP shoot an arrow at ␣]]?"

TRACE

AMBIGUITY

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

[5] If this interpretation seems unlikely, consider "Every Protestant believes in a just God."

Squad helps dog bite victim.
Police begin campaign to run down jaywalkers.
Helicopter powered by human flies.
Once-sagging cloth diaper industry saved by full dumps.
Portable toilet bombed; police have nothing to go on.
Teacher strikes idle kids.
Include your children when baking cookies.
Hospitals are sued by 7 foot doctors.
Milk drinkers are turning to powder.
Safety experts say school bus passengers should be belted.

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. **Lexical ambiguity**, in which a word has more than one meaning, 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 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** refers to a phrase that has multiple parses: "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 for the agent.

LEXICAL AMBIGUITY

SYNTACTIC AMBIGUITY

SEMANTIC AMBIGUITY

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

METONYMY

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

We need to change that to

$$\begin{aligned} x = Chrysler \land e \in Announce(m) \land After(Now, Extent(e)) \\ \land Metonymy(m, x) . \end{aligned}$$

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:

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

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.

METAPHOR

A **metaphor** is another figure of speech, in which a phrase with one literal meaning is used to suggest a different meaning by way of an analogy. Thus, metaphor can be seen as a kind of metonymy where the relation is one of similarity.

DISAMBIGUATION

Disambiguation is the process of recovering the most probable intended meaning of an utterance. In one sense we already have a framework for solving this problem: each rule has a probability associated with it, so the probability of an interpretation is the product of the probabilities of the rules that led to the interpretation. Unfortunately, the probabilities reflect how common the phrases are in the corpus from which the grammar was learned, and thus reflect general knowledge, not specific knowledge of the current situation. To do disambiguation properly, we need to combine four models:

1. The **world model**: the likelihood that a proposition occurs in the world. Given what we know about the world, it is more likely that a speaker who says "I'm dead" means "I am in big trouble" rather than "My life ended, and yet I can still talk."
2. The **mental model**: the likelihood that the speaker forms the intention of communicating a certain fact to the hearer. This approach combines models of what the speaker believes, what the speaker believes the hearer believes, and so on. For example, when a politician says, "I am not a crook," the world model might assign a probability of only 50% to the proposition that the politician is not a criminal, and 99.999% to the proposition that he is not a hooked shepherd's staff. Nevertheless, we select the former interpretation because it is a more likely thing to say.
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.
4. The **acoustic model**: for spoken communication, the likelihood that a particular sequence of sounds will be generated, given that the speaker has chosen a given string of words. Section 23.5 covers speech recognition.

23.4 MACHINE TRANSLATION

Machine translation is the automatic translation of text from one natural language (the source) to another (the target). It was one of the first application areas envisioned for computers (Weaver, 1949), but it is only in the past decade that the technology has seen widespread usage. Here is a passage from page 1 of this book:

> AI is one of the newest fields in science and engineering. Work started in earnest soon after World War II, and the name itself was coined in 1956. Along with molecular biology, AI is regularly cited as the "field I would most like to be in" by scientists in other disciplines.

And here it is translated from English to Danish by an online tool, Google Translate:

> AI er en af de nyeste områder inden for videnskab og teknik. Arbejde startede for alvor lige efter Anden Verdenskrig, og navnet i sig selv var opfundet i 1956. Sammen med molekylær biologi, er AI jævnligt nævnt som "feltet Jeg ville de fleste gerne være i" af forskere i andre discipliner.

For those who don't read Danish, here is the Danish translated back to English. The words that came out different are in italics:

> AI is one of the newest fields *of* science and engineering. Work *began* in earnest *just* after the *Second* World War, and the name itself was *invented* in 1956. *Together* with molecular biology, AI is *frequently mentioned* as ␣ "field I would most like to be in" by *researchers* in other disciplines.

The differences are all reasonable paraphrases, such as *frequently mentioned* for *regularly cited*. The only real error is the omission of the article *the*, denoted by the ␣ symbol. This is typical accuracy: of the two sentences, one has an error that would not be made by a native speaker, yet the meaning is clearly conveyed.

Historically, there have been three main applications of machine translation. *Rough translation*, as provided by free online services, gives the "gist" of a foreign sentence or document, but contains errors. *Pre-edited translation* is used by companies to publish their documentation and sales materials in multiple languages. The original source text is written in a constrained language that is easier to translate automatically, and the results are usually edited by a human to correct any errors. *Restricted-source translation* works fully automatically, but only on highly stereotypical language, such as a weather report.

Translation is difficult because, in the fully general case, it requires in-depth understanding of the text. This is true even for very simple texts—even "texts" of one word. Consider the word "Open" on the door of a store.[6] It communicates the idea that the store is accepting customers at the moment. Now consider the same word "Open" on a large banner outside a newly constructed store. It means that the store is now in daily operation, but readers of this sign would not feel misled if the store closed at night without removing the banner. The two signs use the identical word to convey different meanings. In German the sign on the door would be "Offen" while the banner would read "Neu Eröffnet."

[6] This example is due to Martin Kay.

The problem is that different languages categorize the world differently. For example, the French word "doux" covers a wide range of meanings corresponding approximately to the English words "soft," "sweet," and "gentle." Similarly, the English word "hard" covers virtually all uses of the German word "hart" (physically recalcitrant, cruel) and some uses of the word "schwierig" (difficult). Therefore, representing the meaning of a sentence is more difficult for translation than it is for single-language understanding. An English parsing system could use predicates like $Open(x)$, but for translation, the representation language would have to make more distinctions, perhaps with $Open_1(x)$ representing the "Offen" sense and $Open_2(x)$ representing the "Neu Eröffnet" sense. A representation language that makes all the distinctions necessary for a set of languages is called an **interlingua**.

INTERLINGUA

A translator (human or machine) often needs to understand the actual situation described in the source, not just the individual words. For example, to translate the English word "him," into Korean, a choice must be made between the humble and honorific form, a choice that depends on the social relationship between the speaker and the referent of "him." In Japanese, the honorifics are relative, so the choice depends on the social relationships between the speaker, the referent, and the listener. Translators (both machine and human) sometimes find it difficult to make this choice. As another example, to translate "The baseball hit the window. It broke." into French, we must choose the feminine "elle" or the masculine "il" for "it," so we must decide whether "it" refers to the baseball or the window. To get the translation right, one must understand physics as well as language.

Sometimes there is *no choice* that can yield a completely satisfactory translation. For example, an Italian love poem that uses the masculine "il sole" (sun) and feminine "la luna" (moon) to symbolize two lovers will necessarily be altered when translated into German, where the genders are reversed, and further altered when translated into a language where the genders are the same.[7]

23.4.1 Machine translation systems

All translation systems must model the source and target languages, but systems vary in the type of models they use. Some systems attempt to analyze the source language text all the way into an interlingua knowledge representation and then generate sentences in the target language from that representation. This is difficult because it involves three unsolved problems: creating a complete knowledge representation of everything; parsing into that representation; and generating sentences from that representation.

TRANSFER MODEL

Other systems are based on a **transfer model**. They keep a database of translation rules (or examples), and whenever the rule (or example) matches, they translate directly. Transfer can occur at the lexical, syntactic, or semantic level. For example, a strictly syntactic rule maps English [*Adjective Noun*] to French [*Noun Adjective*]. A mixed syntactic and lexical rule maps French [S_1 "et puis" S_2] to English [S_1 "and then" S_2]. Figure 23.12 diagrams the various transfer points.

[7] Warren Weaver (1949) reports that Max Zeldner points out that the great Hebrew poet H. N. Bialik once said that translation "is like kissing the bride through a veil."

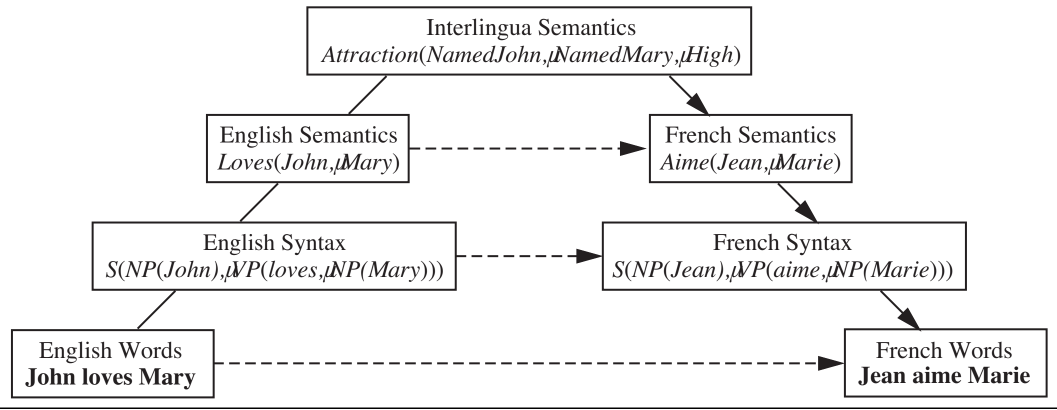


Figure 23.12 The Vauquois triangle: schematic diagram of the choices for a machine translation system (Vauquois, 1968). We start with English text at the top. An interlingua-based system follows the solid lines, parsing English first into a syntactic form, then into a semantic representation and an interlingua representation, and then through generation to a semantic, syntactic, and lexical form in French. A transfer-based system uses the dashed lines as a shortcut. Different systems make the transfer at different points; some make it at multiple points.

23.4.2 Statistical machine translation

Now that we have seen how complex the translation task can be, it should come as no surprise that the most successful machine translation systems are built by training a probabilistic model using statistics gathered from a large corpus of text. This approach does not need a complex ontology of interlingua concepts, nor does it need handcrafted grammars of the source and target languages, nor a hand-labeled treebank. All it needs is data—sample translations from which a translation model can be learned. To translate a sentence in, say, English (e) into French (f), we find the string of words f^* that maximizes

$$f^* = \operatorname*{argmax}_f P(f \mid e) = \operatorname{argmax} P(e \mid f)\, P(f)\ .$$

LANGUAGE MODEL

TRANSLATION MODEL

Here the factor $P(f)$ is the target **language model** for French; it says how probable a given sentence is in French. $P(e|f)$ is the **translation model**; it says how probable an English sentence is as a translation for a given French sentence. Similarly, $P(f \mid e)$ is a translation model from English to French.

Should we work directly on $P(f \mid e)$, or apply Bayes' rule and work on $P(e \mid f)\, P(f)$? In **diagnostic** applications like medicine, it is easier to model the domain in the causal direction: $P(symptoms \mid disease)$ rather than $P(disease \mid symptoms)$. But in translation both directions are equally easy. The earliest work in statistical machine translation did apply Bayes' rule—in part because the researchers had a good language model, $P(f)$, and wanted to make use of it, and in part because they came from a background in speech recognition, which *is* a diagnostic problem. We follow their lead in this chapter, but we note that recent work in statistical machine translation often optimizes $P(f \mid e)$ directly, using a more sophisticated model that takes into account many of the features from the language model.

The language model, $P(f)$, could address any level(s) on the right-hand side of Figure 23.12, but the easiest and most common approach is to build an n-gram model from a French corpus, as we have seen before. This captures only a partial, local idea of French sentences; however, that is often sufficient for rough translation.[8]

BILINGUAL CORPUS

The translation model is learned from a **bilingual corpus**—a collection of parallel texts, each an English/French pair. Now, if we had an infinitely large corpus, then translating a sentence would just be a lookup task: we would have seen the English sentence before in the corpus, so we could just return the paired French sentence. But of course our resources are finite, and most of the sentences we will be asked to translate will be novel. However, they will be composed of **phrases** that we have seen before (even if some phrases are as short as one word). For example, in this book, common phrases include "in this exercise we will," "size of the state space," "as a function of the" and "notes at the end of the chapter." If asked to translate the novel sentence "In this exercise we will compute the size of the state space as a function of the number of actions." into French, we should be able to break the sentence into phrases, find the phrases in the English corpus (this book), find the corresponding French phrases (from the French translation of the book), and then reassemble the French phrases into an order that makes sense in French. In other words, given a source English sentence, e, finding a French translation f is a matter of three steps:

1. Break the English sentence into phrases $e_1, \ldots, e_n$.
2. For each phrase e_i, choose a corresponding French phrase f_i. We use the notation $P(f_i \mid e_i)$ for the phrasal probability that f_i is a translation of e_i.
3. Choose a permutation of the phrases $f_1, \ldots, f_n$. We will specify this permutation in a way that seems a little complicated, but is designed to have a simple probability distribution: For each f_i, we choose a **distortion** d_i, which is the number of words that phrase f_i has moved with respect to f_{i-1}; positive for moving to the right, negative for moving to the left, and zero if f_i immediately follows f_{i-1}.

DISTORTION

Figure 23.13 shows an example of the process. At the top, the sentence "There is a smelly wumpus sleeping in 2 2" is broken into five phrases, $e_1, \ldots, e_5$. Each of them is translated into a corresponding phrase f_i, and then these are permuted into the order f_1, f_3, f_4, f_2, f_5. We specify the permutation in terms of the distortions d_i of each French phrase, defined as

$$d_i = \text{START}(f_i) - \text{END}(f_{i-1}) - 1 \ ,$$

where $\text{START}(f_i)$ is the ordinal number of the first word of phrase f_i in the French sentence, and $\text{END}(f_{i-1})$ is the ordinal number of the last word of phrase f_{i-1}. In Figure 23.13 we see that f_5, "à 2 2," immediately follows f_4, "qui dort," and thus $d_5 = 0$. Phrase f_2, however, has moved one words to the right of f_1, so $d_2 = 1$. As a special case we have $d_1 = 0$, because f_1 starts at position 1 and $\text{END}(f_0)$ is defined to be 0 (even though f_0 does not exist).

Now that we have defined the distortion, d_i, we can define the probability distribution for distortion, $\mathbf{P}(d_i)$. Note that for sentences bounded by length n we have $|d_i| \leq n$, and

[8] For the finer points of translation, n-grams are clearly not enough. Marcel Proust's 4000-page novel *A la récherche du temps perdu* begins and ends with the same word (*longtemps*), so some translators have decided to do the same, thus basing the translation of the final word on one that appeared roughly 2 million words earlier.

so the full probability distribution $\mathbf{P}(d_i)$ has only $2n + 1$ elements, far fewer numbers to learn than the number of permutations, $n!$. That is why we defined the permutation in this circuitous way. Of course, this is a rather impoverished model of distortion. It doesn't say that adjectives are usually distorted to appear after the noun when we are translating from English to French—that fact is represented in the French language model, $P(f)$. The distortion probability is completely independent of the words in the phrases—it depends only on the integer value d_i. The probability distribution provides a summary of the volatility of the permutations; how likely a distortion of $P(d=2)$ is, compared to $P(d=0)$, for example.

We're ready now to put it all together: we can define $P(f, d \mid e)$, the probability that the sequence of phrases f with distortions d is a translation of the sequence of phrases e. We make the assumption that each phrase translation and each distortion is independent of the others, and thus we can factor the expression as

$$P(f, d \mid e) = \prod_i P(f_i \mid e_i)\, P(d_i)$$

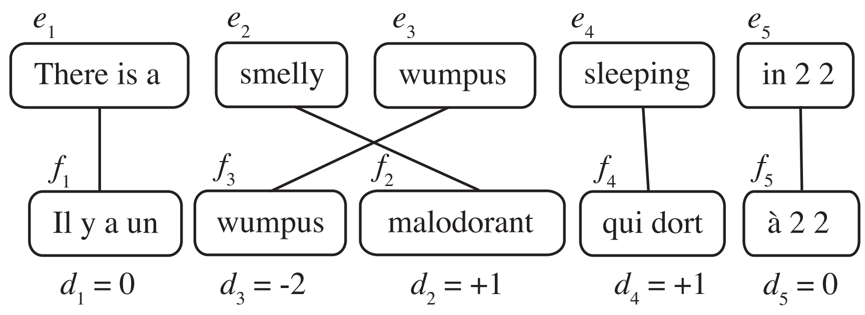


Figure 23.13 Candidate French phrases for each phrase of an English sentence, with distortion (d) values for each French phrase.

That gives us a way to compute the probability $P(f, d \mid e)$ for a candidate translation f and distortion d. But to find the best f and d we can't just enumerate sentences; with maybe 100 French phrases for each English phrase in the corpus, there are 100^5 different 5-phrase translations, and $5!$ reorderings for each of those. We will have to search for a good solution. A local beam search (see page 125) with a heuristic that estimates probability has proven effective at finding a nearly-most-probable translation.

All that remains is to learn the phrasal and distortion probabilities. We sketch the procedure; see the notes at the end of the chapter for details.

HANSARD

1. **Find parallel texts**: First, gather a parallel bilingual corpus. For example, a **Hansard**[9] is a record of parliamentary debate. Canada, Hong Kong, and other countries produce bilingual Hansards, the European Union publishes its official documents in 11 languages, and the United Nations publishes multilingual documents. Bilingual text is also available online; some Web sites publish parallel content with parallel URLs, for

[9] Named after William Hansard, who first published the British parliamentary debates in 1811.

example, `/en/` for the English page and `/fr/` for the corresponding French page. The leading statistical translation systems train on hundreds of millions of words of parallel text and billions of words of monolingual text.

2. **Segment into sentences**: The unit of translation is a sentence, so we will have to break the corpus into sentences. Periods are strong indicators of the end of a sentence, but consider "Dr. J. R. Smith of Rodeo Dr. paid $29.99 on 9.9.09."; only the final period ends a sentence. One way to decide if a period ends a sentence is to train a model that takes as features the surrounding words and their parts of speech. This approach achieves about 98% accuracy.
3. **Align sentences**: For each sentence in the English version, determine what sentence(s) it corresponds to in the French version. Usually, the next sentence of English corresponds to the next sentence of French in a 1:1 match, but sometimes there is variation: one sentence in one language will be split into a 2:1 match, or the order of two sentences will be swapped, resulting in a 2:2 match. By looking at the sentence lengths alone (i.e. short sentences should align with short sentences), it is possible to align them (1:1, 1:2, or 2:2, etc.) with accuracy in the 90% to 99% range using a variation on the Viterbi algorithm. Even better alignment can be achieved by using landmarks that are common to both languages, such as numbers, dates, proper names, or words that we know from a bilingual dictionary have an unambiguous translation. For example, if the 3rd English and 4th French sentences contain the string "1989" and neighboring sentences do not, that is good evidence that the sentences should be aligned together.
4. **Align phrases**: Within a sentence, phrases can be aligned by a process that is similar to that used for sentence alignment, but requiring iterative improvement. When we start, we have no way of knowing that "qui dort" aligns with "sleeping," but we can arrive at that alignment by a process of aggregation of evidence. Over all the example sentences we have seen, we notice that "qui dort" and "sleeping" co-occur with high frequency, and that in the pair of aligned sentences, no phrase other than "qui dort" co-occurs so frequently in other sentences with "sleeping." A complete phrase alignment over our corpus gives us the phrasal probabilities (after appropriate smoothing).
5. **Extract distortions**: Once we have an alignment of phrases we can define distortion probabilities. Simply count how often distortion occurs in the corpus for each distance $d = 0, \pm 1, \pm 2, \ldots$, and apply smoothing.
6. **Improve estimates with EM**: Use expectation–maximization to improve the estimates of $P(f \mid e)$ and $P(d)$ values. We compute the best alignments with the current values of these parameters in the E step, then update the estimates in the M step and iterate the process until convergence.

23.5 SPEECH RECOGNITION

SPEECH RECOGNITION

Speech recognition is the task of identifying a sequence of words uttered by a speaker, given the acoustic signal. It has become one of the mainstream applications of AI—millions of people interact with speech recognition systems every day to navigate voice mail systems, search the Web from mobile phones, and other applications. Speech is an attractive option when hands-free operation is necessary, as when operating machinery.

Speech recognition is difficult because the sounds made by a speaker are ambiguous and, well, noisy. As a well-known example, the phrase "recognize speech" sounds almost the same as "wreck a nice beach" when spoken quickly. Even this short example shows several of the issues that make speech problematic. First, **segmentation**: written words in English have spaces between them, but in fast speech there are no pauses in "wreck a nice" that would distinguish it as a multiword phrase as opposed to the single word "recognize." Second, **coarticulation**: when speaking quickly the "s" sound at the end of "nice" merges with the "b" sound at the beginning of "beach," yielding something that is close to a "sp." Another problem that does not show up in this example is **homophones**—words like "to," "too," and "two" that sound the same but differ in meaning.

SEGMENTATION

COARTICULATION

HOMOPHONES

We can view speech recognition as a problem in most-likely-sequence explanation. As we saw in Section 15.2, this is the problem of computing the most likely sequence of state variables, $\mathbf{x}_{1:t}$, given a sequence of observations $\mathbf{e}_{1:t}$. In this case the state variables are the words, and the observations are sounds. More precisely, an observation is a vector of features extracted from the audio signal. As usual, the most likely sequence can be computed with the help of Bayes' rule to be:

$$\operatorname*{argmax}_{word_{1:t}} P(word_{1:t} \mid sound_{1:t}) = \operatorname*{argmax}_{word_{1:t}} P(sound_{1:t} \mid word_{1:t})P(word_{1:t}) \,.$$

Here $P(sound_{1:t}|word_{1:t})$ is the **acoustic model**. It describes the sounds of words—that "ceiling" begins with a soft "c" and sounds the same as "sealing." $P(word_{1:t})$ is known as the **language model**. It specifies the prior probability of each utterance—for example, that "ceiling fan" is about 500 times more likely as a word sequence than "sealing fan."

ACOUSTIC MODEL

LANGUAGE MODEL

NOISY CHANNEL MODEL

This approach was named the **noisy channel model** by Claude Shannon (1948). He described a situation in which an original message (the *words* in our example) is transmitted over a noisy channel (such as a telephone line) such that a corrupted message (the *sounds* in our example) are received at the other end. Shannon showed that no matter how noisy the channel, it is possible to recover the original message with arbitrarily small error, if we encode the original message in a redundant enough way. The noisy channel approach has been applied to speech recognition, machine translation, spelling correction, and other tasks.

Once we define the acoustic and language models, we can solve for the most likely sequence of words using the Viterbi algorithm (Section 15.2.3 on page 576). Most speech recognition systems use a language model that makes the Markov assumption—that the current state $Word_t$ depends only on a fixed number n of previous states—and represent $Word_t$ as a single random variable taking on a finite set of values, which makes it a Hidden Markov Model (HMM). Thus, speech recognition becomes a simple application of the HMM methodology, as described in Section 15.3—simple that is, once we define the acoustic and language models. We cover them next.

Vowels		Consonants B–N		Consonants P–Z	
Phone	Example	Phone	Example	Phone	Example
[iy]	b**ea**t	[b]	**b**et	[p]	**p**et
[ih]	b**i**t	[ch]	**Ch**et	[r]	**r**at
[eh]	b**e**t	[d]	**d**ebt	[s]	**s**et
[æ]	b**a**t	[f]	**f**at	[sh]	**sh**oe
[ah]	b**u**t	[g]	**g**et	[t]	**t**en
[ao]	b**ough**t	[hh]	**h**at	[th]	**th**ick
[ow]	b**oa**t	[hv]	**h**igh	[dh]	**th**at
[uh]	b**oo**k	[jh]	**j**et	[dx]	bu**tt**er
[ey]	b**ai**t	[k]	**k**ick	[v]	**v**et
[er]	B**er**t	[l]	**l**et	[w]	**w**et
[ay]	b**uy**	[el]	bott**le**	[wh]	**wh**ich
[oy]	b**oy**	[m]	**m**et	[y]	**y**et
[axr]	din**er**	[em]	bott**om**	[z]	**z**oo
[aw]	d**ow**n	[n]	**n**et	[zh]	mea**s**ure
[ax]	**a**bout	[en]	butt**on**		
[ix]	ros**e**s	[ng]	si**ng**		
[aa]	c**o**t	[eng]	wash**ing**	[-]	*silence*

Figure 23.14 The ARPA phonetic alphabet, or **ARPAbet**, listing all the phones used in American English. There are several alternative notations, including an International Phonetic Alphabet (IPA), which contains the phones in all known languages.

23.5.1 Acoustic model

Sound waves are periodic changes in pressure that propagate through the air. When these waves strike the diaphragm of a microphone, the back-and-forth movement generates an electric current. An analog-to-digital converter measures the size of the current—which approximates the amplitude of the sound wave—at discrete intervals called the **sampling rate**. Speech sounds, which are mostly in the range of 100 Hz (100 cycles per second) to 1000 Hz, are typically sampled at a rate of 8 kHz. (CDs and mp3 files are sampled at 44.1 kHz.) The precision of each measurement is determined by the **quantization factor**; speech recognizers typically keep 8 to 12 bits. That means that a low-end system, sampling at 8 kHz with 8-bit quantization, would require nearly half a megabyte per minute of speech.

SAMPLING RATE

QUANTIZATION FACTOR

Since we only want to know what words were spoken, not exactly what they sounded like, we don't need to keep all that information. We only need to distinguish between different speech sounds. Linguists have identified about 100 speech sounds, or **phones**, that can be composed to form all the words in all known human languages. Roughly speaking, a phone is the sound that corresponds to a single vowel or consonant, but there are some complications: combinations of letters, such as "th" and "ng" produce single phones, and some letters produce different phones in different contexts (e.g., the "a" in *rat* and *rate*. Figure 23.14 lists

PHONE

PHONEME

all the phones that are used in English, with an example of each. A **phoneme** is the smallest unit of sound that has a distinct meaning to speakers of a particular language. For example, the "t" in "stick" sounds similar enough to the "t" in "tick" that speakers of English consider them the same phoneme. But the difference is significant in the Thai language, so there they are two phonemes. To represent spoken English we want a representation that can distinguish between different phonemes, but one that need not distinguish the nonphonemic variations in sound: loud or soft, fast or slow, male or female voice, etc.

FRAME

First, we observe that although the sound frequencies in speech may be several kHz, the *changes* in the content of the signal occur much less often, perhaps at no more than 100 Hz. Therefore, speech systems summarize the properties of the signal over time slices called **frames**. A frame length of about 10 milliseconds (i.e., 80 samples at 8 kHz) is short enough to ensure that few short-duration phenomena will be missed. Overlapping frames are used to make sure that we don't miss a signal because it happens to fall on a frame boundary.

FEATURE

MEL FREQUENCY CEPSTRAL COEFFICIENT (MFCC)

Each frame is summarized by a vector of **features**. Picking out features from a speech signal is like listening to an orchestra and saying "here the French horns are playing loudly and the violins are playing softly." We'll give a brief overview of the features in a typical system. First, a Fourier transform is used to determine the amount of acoustic energy at about a dozen frequencies. Then we compute a measure called the **mel frequency cepstral coefficient (MFCC)** or MFCC for each frequency. We also compute the total energy in the frame. That gives thirteen features; for each one we compute the difference between this frame and the previous frame, and the difference between differences, for a total of 39 features. These are continuous-valued; the easiest way to fit them into the HMM framework is to discretize the values. (It is also possible to extend the HMM model to handle continuous mixtures of Gaussians.) Figure 23.15 shows the sequence of transformations from the raw sound to a sequence of frames with discrete features.

PHONE MODEL

We have seen how to go from the raw acoustic signal to a series of observations, $\mathbf{e}_t$. Now we have to describe the (unobservable) states of the HMM and define the transition model, $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-1})$, and the sensor model, $\mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_t)$. The transition model can be broken into two levels: word and phone. We'll start from the bottom: the **phone model** describes

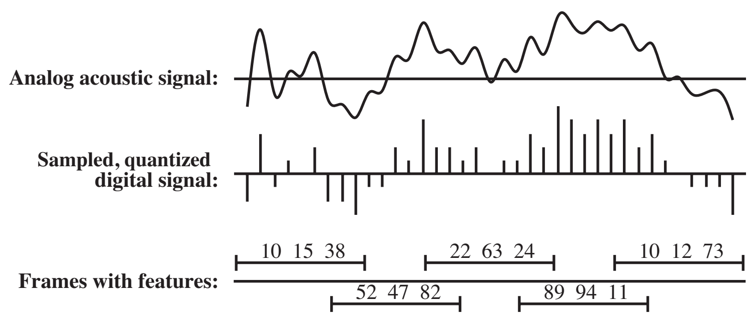
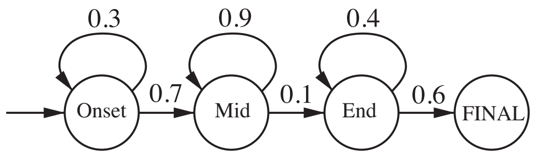


Figure 23.15 Translating the acoustic signal into a sequence of frames. In this diagram each frame is described by the discretized values of three acoustic features; a real system would have dozens of features.

Phone HMM for [m]:



Output probabilities for the phone HMM:

Onset:	Mid:	End:
C_1: 0.5	C_3: 0.2	C_4: 0.1
C_2: 0.2	C_4: 0.7	C_6: 0.5
C_3: 0.3	C_5: 0.1	C_7: 0.4

Figure 23.16 An HMM for the three-state phone [m]. Each state has several possible outputs, each with its own probability. The MFCC feature labels C_1 through C_7 are arbitrary, standing for some combination of feature values.

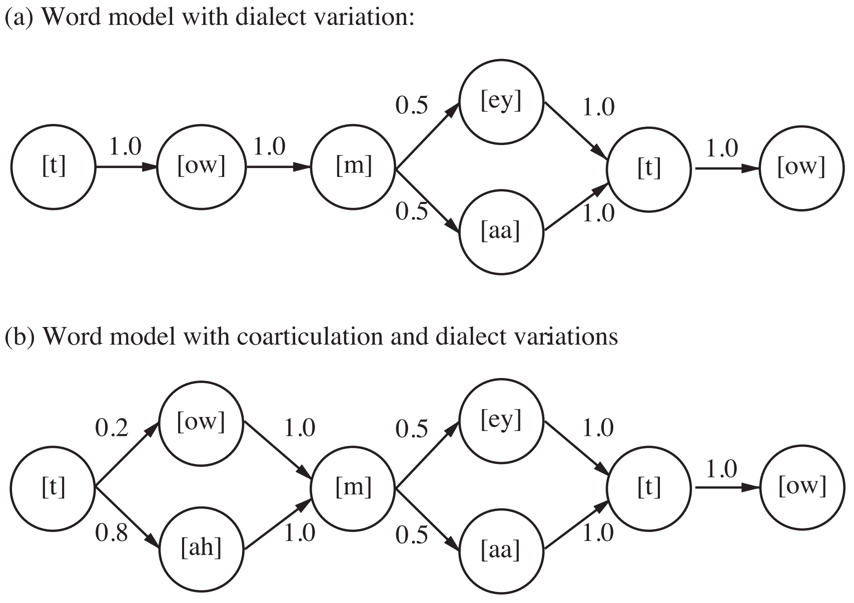


Figure 23.17 Two pronunciation models of the word "tomato." Each model is shown as a transition diagram with states as circles and arrows showing allowed transitions with their associated probabilities. (a) A model allowing for dialect differences. The 0.5 numbers are estimates based on the two authors' preferred pronunciations. (b) A model with a coarticulation effect on the first vowel, allowing either the [ow] or the [ah] phone.

a phone as three states, the onset, middle, and end. For example, the [t] phone has a silent beginning, a small explosive burst of sound in the middle, and (usually) a hissing at the end. Figure 23.16 shows an example for the phone [m]. Note that in normal speech, an average phone has a duration of 50–100 milliseconds, or 5–10 frames. The self-loops in each state allows for variation in this duration. By taking many self-loops (especially in the mid state), we can represent a long "mmmmmmmmmmm" sound. Bypassing the self-loops yields a short "m" sound.

PRONUNCIATION MODEL

In Figure 23.17 the phone models are strung together to form a **pronunciation model** for a word. According to Gershwin (1937), you say [t ow m ey t ow] and I say [t ow m aa t ow]. Figure 23.17(a) shows a transition model that provides for this dialect variation. Each of the circles in this diagram represents a phone model like the one in Figure 23.16.

In addition to dialect variation, words can have **coarticulation** variation. For example, the [t] phone is produced with the tongue at the top of the mouth, whereas the [ow] has the tongue near the bottom. When speaking quickly, the tongue doesn't have time to get into position for the [ow], and we end up with [t ah] rather than [t ow]. Figure 23.17(b) gives a model for "tomato" that takes this coarticulation effect into account. More sophisticated phone models take into account the context of the surrounding phones.

There can be substantial variation in pronunciation for a word. The most common pronunciation of "because" is [b iy k ah z], but that only accounts for about a quarter of uses. Another quarter (approximately) substitutes [ix], [ih] or [ax] for the first vowel, and the remainder substitute [ax] or [aa] for the second vowel, [zh] or [s] for the final [z], or drop "be" entirely, leaving "cuz."

23.5.2 Language model

For general-purpose speech recognition, the language model can be an n-gram model of text learned from a corpus of written sentences. However, spoken language has different characteristics than written language, so it is better to get a corpus of transcripts of spoken language. For task-specific speech recognition, the corpus should be task-specific: to build your airline reservation system, get transcripts of prior calls. It also helps to have task-specific vocabulary, such as a list of all the airports and cities served, and all the flight numbers.

Part of the design of a voice user interface is to coerce the user into saying things from a limited set of options, so that the speech recognizer will have a tighter probability distribution to deal with. For example, asking "What city do you want to go to?" elicits a response with a highly constrained language model, while asking "How can I help you?" does not.

23.5.3 Building a speech recognizer

The quality of a speech recognition system depends on the quality of all of its components—the language model, the word-pronunciation models, the phone models, and the signal-processing algorithms used to extract spectral features from the acoustic signal. We have discussed how the language model can be constructed from a corpus of written text, and we leave the details of signal processing to other textbooks. We are left with the pronunciation and phone models. The *structure* of the pronunciation models—such as the tomato models in

Figure 23.17—is usually developed by hand. Large pronunciation dictionaries are now available for English and other languages, although their accuracy varies greatly. The structure of the three-state phone models is the same for all phones, as shown in Figure 23.16. That leaves the probabilities themselves.

As usual, we will acquire the probabilities from a corpus, this time a corpus of speech. The most common type of corpus to obtain is one that includes the speech signal for each sentence paired with a transcript of the words. Building a model from this corpus is more difficult than building an n-gram model of text, because we have to build a hidden Markov model—the phone sequence for each word and the phone state for each time frame are hidden variables. In the early days of speech recognition, the hidden variables were provided by laborious hand-labeling of spectrograms. Recent systems use expectation–maximization to automatically supply the missing data. The idea is simple: given an HMM and an observation sequence, we can use the smoothing algorithms from Sections 15.2 and 15.3 to compute the probability of each state at each time step and, by a simple extension, the probability of each state–state pair at consecutive time steps. These probabilities can be viewed as *uncertain labels*. From the uncertain labels, we can estimate new transition and sensor probabilities, and the EM procedure repeats. The method is guaranteed to increase the fit between model and data on each iteration, and it generally converges to a much better set of parameter values than those provided by the initial, hand-labeled estimates.

The systems with the highest accuracy work by training a different model for each speaker, thereby capturing differences in dialect as well as male/female and other variations. This training can require several hours of interaction with the speaker, so the systems with the most widespread adoption do not create speaker-specific models.

The accuracy of a system depends on a number of factors. First, the quality of the signal matters: a high-quality directional microphone aimed at a stationary mouth in a padded room will do much better than a cheap microphone transmitting a signal over phone lines from a car in traffic with the radio playing. The vocabulary size matters: when recognizing digit strings with a vocabulary of 11 words (1-9 plus "oh" and "zero"), the word error rate will be below 0.5%, whereas it rises to about 10% on news stories with a 20,000-word vocabulary, and 20% on a corpus with a 64,000-word vocabulary. The task matters too: when the system is trying to accomplish a specific task—book a flight or give directions to a restaurant—the task can often be accomplished perfectly even with a word error rate of 10% or more.

23.6 Summary

Natural language understanding is one of the most important subfields of AI. Unlike most other areas of AI, natural language understanding requires an empirical investigation of actual human behavior—which turns out to be complex and interesting.

- Formal language theory and **phrase structure** grammars (and in particular, **context-free** grammar) are useful tools for dealing with some aspects of natural language. The probabilistic context-free grammar (PCFG) formalism is widely used.

- Sentences in a context-free language can be parsed in $O(n^3)$ time by a **chart parser** such as the **CYK algorithm**, which requires grammar rules to be in **Chomsky Normal Form**.
- A **treebank** can be used to learn a grammar. It is also possible to learn a grammar from an unparsed corpus of sentences, but this is less successful.
- A **lexicalized PCFG** allows us to represent that some relationships between words are more common than others.
- It is convenient to **augment** a grammar to handle such problems as subject–verb agreement and pronoun case. **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.
- **Semantic interpretation** can also be handled by an augmented grammar.
- **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.
- **Machine translation** systems have been implemented using a range of techniques, from full syntactic and semantic analysis to statistical techniques based on phrase frequencies. Currently the statistical models are most popular and most successful.
- **Speech recognition** systems are also primarily based on statistical principles. Speech systems are popular and useful, albeit imperfect.
- Together, machine translation and speech recognition are two of the big successes of natural language technology. One reason that the models perform well is that large corpora are available—both translation and speech are tasks that are performed "in the wild" by people every day. In contrast, tasks like parsing sentences have been less successful, in part because no large corpora of parsed sentences are available "in the wild" and in part because parsing is not useful in and of itself.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

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, ca. 350 B.C.) studying Shastric 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. Peter Naur 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 Shieber (1987).

ATTRIBUTE GRAMMAR

Probabilistic context-free grammars were investigated by Booth (1969) and Salomaa (1969). Other algorithms for PCFGs are presented in the excellent short monograph by

Charniak (1993) and the excellent long textbooks by Manning and Schütze (1999) and Jurafsky and Martin (2008). Baker (1979) introduces the inside–outside algorithm for learning a PCFG, and Lari and Young (1990) describe its uses and limitations. Stolcke and Omohundro (1994) show how to learn grammar rules with Bayesian model merging; Haghighi and Klein (2006) describe a learning system based on prototypes.

Lexicalized PCFGs (Charniak, 1997; Hwa, 1998) combine the best aspects of PCFGs and n-gram models. Collins (1999) describes PCFG parsing that is lexicalized with head features. Petrov and Klein (2007a) show how to get the advantages of lexicalization without actual lexical augmentations by learning specific syntactic categories from a treebank that has general categories; for example, the treebank has the category NP, from which more specific categories such as NP_O and NP_S can be learned.

There have been many attempts to write formal grammars of natural languages, both in "pure" linguistics and in computational linguistics. There are several comprehensive but informal grammars of English (Quirk *et al.*, 1985; McCawley, 1988; Huddleston and Pullum, 2002). 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** (Clark and Curran, 2004), in which there can be as few as two grammar rules, or with **dependency grammar** (Smith and Eisner, 2008; Kübler *et al.*, 2009) in which there are no syntactic categories, only relations between words. Sleator and Temperley (1993) describe a dependency parser. Paskin (2001) shows that a version of dependency grammar is easier to learn than PCFGs.

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; Earley, 1970; Graham *et al.*, 1980). 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. Klein and Manning (2003) describe A* parsing, and Pauls and Klein (2009) extend that to *K*-best A* parsing, in which the result is not a single parse but the *K* best.

Leading parsers today include those by Petrov and Klein (2007b), which achieved 90.6% accuracy on the Wall Street Journal corpus, Charniak and Johnson (2005), which achieved 92.0%, and Koo *et al.* (2008), which achieved 93.2% on the Penn treebank. These numbers are not directly comparable, and there is some criticism of the field that it is focusing too narrowly on a few select corpora, and perhaps overfitting on them.

Formal semantic interpretation of natural languages originates within philosophy and formal logic, particularly Alfred Tarski's (1935) work on the semantics of formal languages. Bar-Hillel (1954) 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. Richard Montague's essay "English as a formal language" (1970) is a kind of manifesto for the logical analysis of language, but the books by Dowty *et al.* (1991) and Portner and Partee (2002) are more readable.

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 from the moon by the Apollo program. Roger Schank and his students built a series of programs (Schank and Abelson, 1977; Schank and Riesbeck, 1981) that all had the task of understanding language. Modern approaches to semantic interpretation usually assume that the mapping from syntax to semantics will be learned from examples (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005).

Hobbs *et al.* (1993) describes a quantitative nonprobabilistic framework for interpretation. More recent work follows an explicitly probabilistic framework (Charniak and Goldman, 1992; Wu, 1993; Franz, 1996). In linguistics, optimality theory (Kager, 1999) is based on the idea of building soft constraints into the grammar, giving a natural ranking to interpretations (similar to a probability distribution), 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.

Nunberg (1979) outlines a formal model of metonymy. Lakoff and Johnson (1980) give an engaging analysis and catalog of common metaphors in English. Martin (1990) and Gibbs (2006) offer computational models of metaphor interpretation.

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. Prominent linguists, such as Chomsky (1957) and Pinker (2003), 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 says that children aren't given enough input to learn a CFG, so they must already "know" the grammar and be merely tuning some of its parameters. While this argument continues to hold sway throughout much of Chomskyan 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), the sequence learning of Nevill-Manning and Witten (1997), and the remarkable Ph.D. theses of Schütze (1995) and de Marcken (1996). There is an annual International Conference on Grammatical Inference (ICGI). It is possible to learn other grammar formalisms, such as regular languages (Denis, 2001) and finite state automata (Parekh and Honavar, 2001). Abney (2007) is a textbook introduction to semi-supervised learning for language models.

UNIVERSAL GRAMMAR

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 Penn Treebank (Marcus *et al.*, 1993) provides parse trees for a 3-million-word corpus of English. Charniak (1996) and Klein and Manning (2001) discuss parsing with treebank grammars. The British National Corpus (Leech *et al.*, 2001) contains 100 million words, and the World Wide Web contains several trillion words; (Brants *et al.*, 2007) describe n-gram models over a 2-trillion-word Web corpus.

In the 1930s Petr Troyanskii applied for a patent for a "translating machine," but there were no computers available to implement his ideas. In March 1947, the Rockefeller Foundation's Warren Weaver wrote to Norbert Wiener, suggesting that machine translation might be possible. Drawing on work in cryptography and information theory, Weaver wrote, "When I look at an article in Russian, I say: 'This is really written in English, but it has been coded in strange symbols. I will now proceed to decode.'" For the next decade, the community tried to decode in this way. IBM exhibited a rudimentary system in 1954. Bar-Hillel (1960) describes the enthusiasm of this period. However, the U.S. government subsequently reported (ALPAC, 1966) that "there is no immediate or predictable prospect of useful machine translation." However, limited work continued, and starting in the 1980s, computer power had increased to the point where the ALPAC findings were no longer correct.

The basic statistical approach we describe in the chapter is based on early work by the IBM group (Brown *et al.*, 1988, 1993) and the recent work by the ISI and Google research groups (Och and Ney, 2004; Zollmann *et al.*, 2008). A textbook introduction on statistical machine translation is given by Koehn (2009), and a short tutorial by Kevin Knight (1999) has been influential. Early work on sentence segmentation was done by Palmer and Hearst (1994). Och and Ney (2003) and Moore (2005) cover bilingual sentence alignment.

The prehistory of speech recognition began in the 1920s with Radio Rex, a voice-activated toy dog. Rex jumped out of his doghouse in response to the word "Rex!" (or actually almost any sufficiently loud word). Somewhat more serious work began after World War II. At AT&T Bell Labs, a system was built for recognizing isolated digits (Davis *et al.*, 1952) by means of simple pattern matching of acoustic features. Starting in 1971, the Defense Advanced Research Projects Agency (DARPA) of the United States Department of Defense funded four competing five-year projects to develop high-performance speech recognition systems. The winner, and the only system to meet the goal of 90% accuracy with a 1000-word vocabulary, was the HARPY system at CMU (Lowerre and Reddy, 1980). The final version of HARPY was derived from a system called DRAGON built by CMU graduate student James Baker (1975); DRAGON was the first to use HMMs for speech. Almost simultaneously, Jelinek (1976) at IBM had developed another HMM-based system. Recent years have been characterized by steady incremental progress, larger data sets and models, and more rigorous competitions on more realistic speech tasks. In 1997, Bill Gates predicted, "The PC five years from now—you won't recognize it, because speech will come into the interface." That didn't quite happen, but in 2008 he predicted "In five years, Microsoft expects more Internet searches to be done through speech than through typing on a keyboard." History will tell if he is right this time around.

Several good textbooks on speech recognition are available (Rabiner and Juang, 1993; Jelinek, 1997; Gold and Morgan, 2000; Huang *et al.*, 2001). The presentation in this chapter drew on the survey by Kay, Gawron, and Norvig (1994) and on the textbook by Jurafsky and Martin (2008). Speech recognition research is published in *Computer Speech and Language*, *Speech Communications*, and the IEEE *Transactions on Acoustics, Speech, and Signal Processing* and at the DARPA Workshops on Speech and Natural Language Processing and the Eurospeech, ICSLP, and ASRU conferences.

Ken Church (2004) shows that natural language research has cycled between concentrating on the data (empiricism) and concentrating on theories (rationalism). The linguist John Firth (1957) proclaimed "You shall know a word by the company it keeps," and linguistics of the 1940s and early 1950s was based largely on word frequencies, although without the computational power we have available today. Then Noam (Chomsky, 1956) showed the limitations of finite-state models, and sparked an interest in theoretical studies of syntax, disregarding frequency counts. This approach dominated for twenty years, until empiricism made a comeback based on the success of work in statistical speech recognition (Jelinek, 1976). Today, most work accepts the statistical framework, but there is great interest in building statistical models that consider higher-level models, such as syntactic trees and semantic relations, not just sequences of words.

Work on applications of language processing is presented at the biennial Applied Natural Language Processing conference (ANLP), the conference on Empirical Methods in Natural Language Processing (EMNLP), and the journal *Natural Language Engineering*. A broad range of NLP work appears in the journal *Computational Linguistics* and its conference, ACL, and in the Computational Linguistics (COLING) conference.

EXERCISES

23.1 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.

23.2 An *HMM grammar* is essentially a standard HMM whose state variable is N (nonterminal, with values such as Det, $Adjective$, $Noun$ and so on) and whose evidence variable is W (word, with values such as is, $duck$, and so on). The HMM model includes a prior $\mathbf{P}(N_0)$, a transition model $\mathbf{P}(N_{t+1}|N_t)$, and a sensor model $\mathbf{P}(W_t|N_t)$. Show that every HMM grammar can be written as a PCFG. [Hint: start by thinking about how the HMM prior can be represented by PCFG rules for the sentence symbol. You may find it helpful to illustrate for the particular HMM with values A, B for N and values x, y for W.]

23.3 Consider the following PCFG for simple verb phrases:

$0.1 : VP \rightarrow Verb$
$0.2 : VP \rightarrow Copula\ Adjective$
$0.5 : VP \rightarrow Verb\ the\ Noun$
$0.2 : VP \rightarrow VP\ Adverb$
$0.5 : Verb \rightarrow is$
$0.5 : Verb \rightarrow shoots$
$0.8 : Copula \rightarrow is$
$0.2 : Copula \rightarrow seems$
$0.5 : Adjective \rightarrow$ **unwell**
$0.5 : Adjective \rightarrow$ **well**
$0.5 : Adverb \rightarrow$ **well**
$0.5 : Adverb \rightarrow$ **badly**
$0.6 : Noun \rightarrow$ **duck**
$0.4 : Noun \rightarrow$ **well**

a. Which of the following have a nonzero probability as a VP? (i) shoots the duck well well well (ii) seems the well well (iii) shoots the unwell well badly

b. What is the probability of generating "is well well"?

c. What types of ambiguity are exhibited by the phrase in (b)?

d. Given any PCFG, is it possible to calculate the probability that the PCFG generates a string of exactly 10 words?

23.4 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.

23.5 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.

23.6 Consider the sentence "Someone walked slowly to the supermarket" and a lexicon consisting of the following words:

$Pronoun \rightarrow$ **someone**	$Verb \rightarrow$ **walked**
$Adv \rightarrow$ **slowly**	$Prep \rightarrow$ **to**
$Article \rightarrow$ **the**	$Noun \rightarrow$ **supermarket**

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

(A):	(B):	(C):
$S \rightarrow NP \;\; VP$	$S \rightarrow NP \;\; VP$	$S \rightarrow NP \;\; VP$
$NP \rightarrow Pronoun$	$NP \rightarrow Pronoun$	$NP \rightarrow Pronoun$
$NP \rightarrow Article \;\; Noun$	$NP \rightarrow Noun$	$NP \rightarrow Article \;\; NP$
$VP \rightarrow VP \;\; PP$	$NP \rightarrow Article \;\; NP$	$VP \rightarrow Verb \;\; Adv$
$VP \rightarrow VP \;\; Adv \;\; Adv$	$VP \rightarrow Verb \;\; Vmod$	$Adv \rightarrow Adv \;\; Adv$
$VP \rightarrow Verb$	$Vmod \rightarrow Adv \;\; Vmod$	$Adv \rightarrow PP$
$PP \rightarrow Prep \;\; NP$	$Vmod \rightarrow Adv$	$PP \rightarrow Prep \;\; NP$
$NP \rightarrow Noun$	$Adv \rightarrow PP$	$NP \rightarrow Noun$
	$PP \rightarrow Prep \;\; NP$	

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 include some new lexical entries (which you should define). Suggest ways to improve each grammar to avoid generating the non-English sentences.

23.7 Collect some examples of time expressions, such as "two o'clock," "midnight," and "12:46." Also think up some examples that are ungrammatical, such as "thirteen o'clock" or "half past two fifteen." Write a grammar for the time language.

23.8 In this exercise you will transform $\mathcal{E}_0$ into Chomsky Normal Form (CNF). There are five steps: (a) Add a new start symbol, (b) Eliminate ϵ rules, (c) Eliminate multiple words on right-hand sides, (d) Eliminate rules of the form $(X \rightarrow Y)$, (e) Convert long right-hand sides into binary rules.

a. The start symbol, S, can occur only on the left-hand side in CNF. Add a new rule of the form $S' \rightarrow S$, using a new symbol S'.

b. The empty string, ϵ cannot appear on the right-hand side in CNF. $\mathcal{E}_0$ does not have any rules with ϵ, so this is not an issue.

c. A word can appear on the right-hand side in a rule only of the form ($X \rightarrow$ *word*). Replace each rule of the form ($X \rightarrow \ldots$*word*$\ldots$) with ($X \rightarrow \ldots\; W' \ldots$) and ($W' \rightarrow$ *word*), using a new symbol W'.

d. A rule $(X \rightarrow Y)$ is not allowed in CNF; it must be $(X \rightarrow Y \; Z)$ or ($X \rightarrow$ *word*). Replace each rule of the form $(X \rightarrow Y)$ with a set of rules of the form $(X \rightarrow \ldots)$, one for each rule $(Y \rightarrow \ldots)$, where $(\ldots)$ indicates one or more symbols.

e. Replace each rule of the form $(X \rightarrow Y \; Z \ldots)$ with two rules, $(X \rightarrow Y \; Z')$ and $(Z' \rightarrow Z \ldots)$, where Z' is a new symbol.

Show each step of the process and the final set of rules.

23.9 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 ungrammatical sentences such as "I smells the wumpus."

23.10 Consider the following PCFG:

$S \rightarrow NP\ VP$ [1.0]
$NP \rightarrow Noun$ [0.6] | $Pronoun$ [0.4]
$VP \rightarrow Verb\ NP$ [0.8] | $Modal\ Verb$ [0.2]

$Noun \rightarrow$ **can** [0.1] | **fish** [0.3] | ...
$Pronoun \rightarrow$ **I** [0.4] | ...
$Verb \rightarrow$ **can** [0.01] | **fish** [0.1] | ...
$Modal \rightarrow$ **can** [0.3] | ...

The sentence "I can fish" has two parse trees with this grammar. Show the two trees, their prior probabilities, and their conditional probabilities, given the sentence.

23.11 An augmented context-free grammar can represent languages that a regular context-free grammar cannot. Show an augmented context-free grammar for the language $a^n b^n c^n$. The allowable values for augmentation variables are 1 and $\textsc{Successor}(n)$, where n is a value. The rule for a sentence in this language is

$$S(n) \rightarrow A(n)\ B(n)\ C(n)\ .$$

Show the rule(s) for each of A, B, and C.

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

23.13 Consider the following sentence (from *The New York Times,* July 28, 2008):

> Banks struggling to recover from multibillion-dollar loans on real estate are curtailing loans to American businesses, depriving even healthy companies of money for expansion and hiring.

a. Which of the words in this sentence are lexically ambiguous?
b. Find two cases of syntactic ambiguity in this sentence (there are more than two.)
c. Give an instance of metaphor in this sentence.
d. Can you find semantic ambiguity?

23.14 Without looking back at Exercise 23.1, 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 things or too many? Why?

23.15 Select five sentences and submit them to an online translation service. Translate them from English to another language and back to English. Rate the resulting sentences for grammaticality and preservation of meaning. Repeat the process; does the second round of iteration give worse results or the same results? Does the choice of intermediate language make a difference to the quality of the results? If you know a foreign language, look at the translation of one paragraph into that language. Count and describe the errors made, and conjecture why these errors were made.

23.16 The D_i values for the sentence in Figure 23.13 sum to 0. Will that be true of every translation pair? Prove it or give a counterexample.

23.17 (Adapted from Knight (1999).) Our translation model assumes that, after the phrase translation model selects phrases and the distortion model permutes them, the language model can unscramble the permutation. This exercise investigates how sensible that assumption is. Try to unscramble these proposed lists of phrases into the correct order:

a. have, programming, a, seen, never, I, language, better

b. loves, john, mary

c. is the, communication, exchange of, intentional, information brought, by, about, the production, perception of, and signs, from, drawn, a, of, system, signs, conventional, shared

d. created, that, we hold these, to be, all men, truths, are, equal, self-evident

Which ones could you do? What type of knowledge did you draw upon? Train a bigram model from a training corpus, and use it to find the highest-probability permutation of some sentences from a test corpus. Report on the accuracy of this model.

23.18 Calculate the most probable path through the HMM in Figure 23.16 for the output sequence $[C_1, C_2, C_3, C_4, C_4, C_6, C_7]$. Also give its probability.

23.19 We forgot to mention that the text in Exercise 23.1 is entitled “Washing Clothes.” Reread the text and answer the questions in Exercise 23.14. Did you do better this time? Bransford and Johnson (1973) used this text in a controlled experiment and found that the title helped significantly. What does this tell you about how language and memory works?