25 Computational Linguistics

RICHARD SPROAT, CHRISTER SAMUELSSON, JENNIFER CHU-CARROLL, and BOB CARPENTER

The field of computational linguistics is as diverse as linguistics itself, so giving a thorough overview of the entire field in the short space available for this chapter is essentially impossible. We have therefore chosen to focus on four relatively popular areas of inquiry:

- syntactic parsing;
- discourse analysis;
- computational morphology and phonology;
- corpus-based methods.

The order of presentation is motivated by historical considerations. Parsing and discourse analysis have had the longest continuous history of investigation, and are therefore presented first. Computational morphology and phonology only really began to grow as a separate discipline in the mid-1980s. Corpus-based approaches were, in fact, investigated as early as the 1960s (e.g., by Zellig Harris (1970)), but the field fell into disrepute until the late 1980s, since which time there has been a renaissance of work in this area.

1 Parsing

Parsing is the act of determining the "syntactic structure" of a sentence. Although syntactic theories differ on their notions of structure, the goal of such structure is typically to represent "who did what to whom" in the sentence. Any natural language processing system that needs to produce an interpretation from the utterance that is deeper than a bag of keywords thus involves some form

of parsing (see section 1.5 for examples of existing practical applications of parsing).

Parsing typically involves tagging the words with an appropriate syntactic category and determining their relationships to each other. More often than not, words are grouped into phrase-like constituents, which are subsequently arranged into clauses and sentences.

1.1 Phrase structure grammars

For the purposes of this section, we will restrict our attention to the most widely studied form of grammars for computational purposes: *context-free phrase structure grammars*. We will assume familiarity with phrase structure grammars and the notion of phrase marker. Consider the following simple context-free grammar for verb phrases and noun phrases with prepositional modifiers:

VP	→	TV NP	TV	→	*saw, knows, liked, met*
VP	→	VP PP	NP	→	*Sandy, Chris*
NP	→	Det N	N	→	*kid, telescope, saw, field*
NP	→	NP PP	PP	→	*outside, somewhere*
PP	→	P NP	P	→	*in, on, near, with*
			Det	→	*the, a, every*

For any possible sequence of words, the grammar determines what, if any, valid *phrase structures* it has. For example, the expression *the kid* has exactly one analysis in this grammar, namely [NP [Det *the*] [N *kid*]]. According to this grammar, the expressions *kid the* and *the dog* are also ungrammatical; the former because there is no rule $C \rightarrow$ N Det (where C is any category) in the grammar, the latter because there is no lexical entry for *dog*. Some expressions have multiple analyses, such as the classic example *saw the kid with the telescope*, which has the two (abbreviated) structures [VP [VP *saw the kid*] [PP *with the telescope*]] and [VP *saw* [NP [NP *the kid*] [PP *with the telescope*]]]. This example contains a so-called *structural ambiguity* between the situation in which the prepositional phrase *with the telescope* modifies the noun phrase *the kid* and one in which it modifies the verb phrase *saw the kid*.

1.2 Parsing as search

Almost all parsers involve some notion of searching for possible analyses for a given sequence of words. *Top-down* parsers are goal driven, and begin their search from the answer (the top of a tree) and work down to the actual expression input to the parser. *Bottom-up* parsers are *data driven*, beginning from lexical categories from the input and combining them into larger phrases.

1.2.1 Bottom up: shift-reduce parsing

One standard approach to bottom-up parsing is the shift-reduce framework, which allows two operations: shifting lexical material (i.e., replacing words with their grammatical categories) and reducing sequences of categories based on rule applications. Parsing begins from the input sequence and the state of the parse during parsing is represented by two sequences: the sequence of words remaining to parse and the sequence of categories already found. For instance, the prepositional phrase *with the telescope* would invoke the following parser steps.

Words	Categories	Operation	Rule
with the telescope	found	initialize	
the telescope	P	shift	P → *with*
telescope	P Det	shift	Det → *the*
	P Det N	shift	N → *telescope*
	P NP	reduce	NP → Det N
	PP	reduce	PP → P NP

The parser is initialized with the string and allows two operations. First, the first word on the list of words can be replaced with one of its lexical entries through shifting, as in the first two steps after initialization above.

$$\frac{\mathbf{w}, u_1, \ldots, u_n \quad C_1, \ldots, C_m}{u_1, \ldots, u_n \quad C_1, \ldots, C_m, \mathbf{D}} \qquad [\mathbf{D} \rightarrow \mathbf{w} \text{ in lexicon}]$$

In general, our rules operate on sequences of words to parse and categories found, returning the same type of result. The first half of the input to the above rule is a sequence of words $\mathbf{w}, u_1, \ldots, u_n$, the first of which is the word $\mathbf{w}$ we are going to lexically rewrite. The second half of the input is the current set of categories found, $C_1, \ldots, C_n$. The result of the application of the rule is the sequence of categories $u_1, \ldots, u_n$ with the first element $\mathbf{w}$ from the input removed, along with the sequence of categories $C_1, \ldots, C_n, \mathbf{D}$ where the lexical entry category $\mathbf{D}$ has been added to the list of categories in the input.

The second operation allows the reduction of the rightmost sequence of categories by means of a rule, as in the last two steps in the derivation above.

$$\frac{w_1, \ldots, w_n \quad D_1, \ldots, D_m, \mathbf{C_1}, \ldots, \mathbf{C_k}}{w_1, \ldots, w_n \quad D_1, \ldots, D_m, \mathbf{C_0}} \qquad [\mathbf{C_0} \rightarrow \mathbf{C_1}, \ldots, \mathbf{C_k} \text{ in grammar}]$$

In this rule, the sequence of words $w_1, \ldots, w_n$ is unchanged. A final subsequence $C_1, \ldots, C_k$ of categories from the input sequence of previously derived categories $D_1, \ldots, D_m, \mathbf{C_1}, \ldots, \mathbf{C_k}$, is rewritten according to a grammar rule. Thus because $C_0 \rightarrow C_1, \ldots, C_k$ is a rule in the grammar, if we have found the categories $C_1, \ldots, C_k$ in the input, we can replace them with their mother

category C_0 from the rule. Note that we can restrict our application of grammar rules to final subsequences without losing any parses.

In cases of ambiguity, there will be more than one alternative expansion at some point in the parsing process. For instance, consider the expression *saw Sandy outside*, which is a minimal length string displaying PP attachment ambiguity. Here are the two derivations.

saw Sandy outside		*saw Sandy outside*	
Sandy outside	TV	*Sandy outside*	TV
outside	TV NP	*outside*	TV NP
	TV NP PP	*outside*	VP
	TV NP		VP PP
	VP		VP

The critical decision here is made after the lexical categories for *saw* and *Sandy* have been shifted, when it must be decided whether to shift the preposition or complete the verb phrase. The parse trees can be straightforwardly reconstructed from the steps taken by the parser.

1.2.2 Top-down: recursive-descent parsing

A standard approach to top-down parsing is by means of *recursive descent*. This strategy involves recursively expanding categories until they match lexical material. This can be modeled procedurally in much the same way as bottom-up parsing, using a list of categories and of lexical material. The difference is that the list of categories are categories that have not been found yet, in contrast to the shift-reduce parser, where the categories list constituents which have already been found.

Words	Categories needed	Operation	Rule
with the telescope	PP	initialize	
with the telescope	P NP	expand	PP → P NP
the telescope	NP	lex	P → *with*
the telescope	Det N	expand	NP → Det N
telescope	N	lex	Det → *the*
		lex	N → *telescope*

With top-down parsing, initialization involves beginning with the category being sought, which in the case above, is PP. Although it may look the same, this is totally different from the representation in bottom-up parsing, because the categories involved in a step of top-down parsing have *not* been found.

Two rules then define the search. The first allows lexical matching of the first category being sought against the first of the remaining words.

$$\frac{\mathbf{w}, u_1, \ldots, u_n \quad \mathbf{D}, C_1, \ldots, C_m}{u_1, \ldots, u_n \quad C_1, \ldots, C_m} \qquad [\mathbf{D} \rightarrow \mathbf{w} \text{ in lexicon}]$$

Thus if the first word we have on our list of words to be processed is **w** and the current category we are seeking is **D**, then if there is a lexical entry of category **D** for **w**, we can remove the word from the input sequence and the category from the sequence of categories being sought.

The second scheme allows a category being sought to be expanded according to a grammar rule.

$$\frac{w_1, \ldots, w_n \quad \mathbf{C_0}, D_1, \ldots, D_m}{w_1, \ldots, w_n \quad \mathbf{C_1}, \ldots, \mathbf{C_k}, D_1, \ldots, D_m} \qquad [\mathbf{C_0} \rightarrow \mathbf{C_1}, \ldots, \mathbf{C_k} \text{ in grammar}]$$

This simply says that if there is a grammar rule $C_0 \rightarrow C_1, \ldots, C_k$ and the first category we are seeking is C_0, then we can remove the C_0 from the list of categories being sought and replace it with the sequence of categories $C_1, \ldots, C_k$. As with bottom-up parsing, we can restrict the position of the rule application without loss of generality. This time, we require the top-down rewriting to be applied to the first category on the sequence of categories being sought.

Consider the resolution of attachment ambiguity by means of the top-down processor.

saw Sandy outside	VP	*saw Sandy outside*	VP
saw Sandy outside	VP PP	*saw Sandy outside*	TV NP
saw Sandy outside	TV NP PP	*Sandy outside*	NP
Sandy outside	NP PP	*Sandy outside*	NP PP
outside	PP	*outside*	PP

The criticial decision here is made before the verb phrase is expanded; we must decide whether to expand it as a TV-NP sequence or as a VP-PP sequence, which decides the attachment.

1.2.3 Complexity of search-based parsing

In general, a search-based parser must explore the entire search space in order to find all parses (or to reject a string as ungrammatical). The problem with this is that the search space suffers a combinatorial explosion as the length of the string grows. Consider a string of k prepositional phrases each consisting of a preposition, determiner, and noun (for $k = 4$, an instance is *beside the dog near the radiator in the house by the street*). The problem is that there are more than $2^{(k-2)}$ valid structural analyses of a sequence k preposition, determiner, noun sequences. This means that both search-based parsing strategies will require at least $2^{(k-2)}$ steps to analyze a sequence of $3k$ words in the worst case. The actual growth in number of analyses follows the *Catalan Numbers*:

$$b_n = \frac{1}{n+1}\binom{2n}{n}$$

which give the number b_n of binary bracketings (or equivalently parse tree structures), for a string of length n.

Whether top-down or bottom-up parsers will be more efficient for a particular application depends on the grammar and input strings. Ambiguity in top-down parsing stems from having many rules with the same mother category, whereas ambiguity in bottom-up parsing arises from a high degree of lexical ambiguity or by having a high degree of ambiguity in the daughters of rules.

1.3 Parsing as dynamic programming

Dynamic programming is a standard computational technique for keeping track of subcomputations that have already been performed. In the context of parsing, it will amount to keeping track of all possible analyses for each subsequence of the input expression. The key insight is that we are dealing with context-free languages. That is, the possible categories that we can assign to a sequence of words does not depend on the context in which it is found. Thus no matter how many ways there are to analyze a sequence $(\text{P Det N})^k$, the result is always a prepositional phrase and the internal structure will not influence how it can combine with other constituents in an analysis.

1.3.1 The Cocke–Kasami–Younger parser

The first and simplest parser involving dynamic programming is the Cocke–Kasami–Younger (CKY) parser (Younger 1967). The principle here is to iterate over ever-larger subsequences of the input, computing all possible analyses of the subsequence and recording them for future analysis. CKY parsing is only defined for grammars all of whose rules are binary branching, but there are other parsers based on dynamic programming which work over arbitrary grammars. For simplicity, we will restrict our attention to grammars all of whose rules are binary branching. A table representing all such analyses for the string *saw Sandy outside* is as follows, where the vertical axis represents starting positions and the horizontal axis represents ending positions.

	1	2	3
1	TV, NP	VP	VP
2		NP	NP
3			PP

For instance, the NP entry at (2, 3) indicates that there is a noun phrase spanning from the second to the third word, which is the string *Sandy outside*. Similarly, the PP entry from (3, 3), indicates that the substring *outside* can be analyzed as a prepositional phrase. The fact that there are two entries at (1, 1), indicates that the string *saw* is ambiguous between a TV and NP.

The method of constructing a matrix representation as above is to work from small spans outward. This ensures that when we come to analyze a longer sequence, all of the subsequences have been fully analyzed. We can work left to right, and fill out the entries in the table above in the order (1, 1), (2, 2), (3, 3), (1, 2), (2, 3), (1, 3). To fill in an entry (n, n) in the table, we simply fill in all possible lexical entries for the nth word in the input. To fill in an entry (n, m) for $m > n$, we inspect all break points k such that $m \leq k < n$. This enumerates all the ways a phrase spanning from n to m could be subdivided into two subphrases. For each such k, we consider all possible ways to combine the categories spanning the interval (m, k) with the categories spanning the interval $(k + 1, n)$ given the grammar rules. If there is a category C_1 spanning (m, k) and a C_2 spanning $(k + 1, n)$, and there is a grammar rule $C_0 \rightarrow C_1C_2$, then there will be a category C_0 spanning (m, n). For instance, to fill in (1, 3) above, we consider the break points 1 and 2. For (1, 1) we have a TV and an NP, and for (2, 3) we have an NP, and we can combine TV-NP to produce a VP, so VP is included in the categories in (1, 3), but there is no way to combine an NP-NP sequence (in this grammar). The fact that we have an NP spanning (1, 1) and an NP spanning (2, 3) does not add any further analyses. The VP spanning (1, 2) and the PP spanning (3, 3) entails that we have a VP spanning (1, 3). This represents the second way in which the string could be analyzed.

It is fairly easy to see that the amount of work carried out under the CKY approach is bounded by the number of boxes in the matrix and the amount of work that could be done for each box. For an input of length n there are n^2 possible boxes. For each box, we have to consider all possible split points of which there are at most n, which leads to n^3 points at which the grammar is consulted. The amount of work to do at each split point is bounded by the number of categories and rules in the grammar.

So what happens to our sequence of k prepositional phrases? Consider *in the box near the chair under the door*, for $k = 3$. In addition to lexical entries, we wind up with PP entries at (1, 3), (4, 6), (7, 9), (1, 6), (4, 9), and (1, 9), with NP entries at (2, 3), (5, 6), (8, 9), (2, 6), (5, 9), and (2, 9).

1.4 Scaling up

We now know that we can build a parser for context-free grammars that takes at most n^3 steps for an input sequence of length n. For wide coverage applications, this is still prohibitive. For instance, the Penn Treebank (Marcus et al. 1993), a corpus of one million words drawn from the *Wall Street Journal* and analyzed by hand, already involves an implicit grammar of roughly 8,000 rules and a lexicon of tens of thousands of words and is by no means complete. Sentences average just over 21 words each. These sentences, when fed back into the parser, result in hundreds of thousands and often millions of

well-formed substrings. Current "large-scale" grammars for linguistic theories such as HPSG and LFG run to hundreds of pages of coding for the lexicon, lexical rules, and grammar.

1.5 Applications

Parsing has been successfully applied in a number of domains. State of the art machine translation systems employ parsing to derive representations of the input that are sufficient for transfer from the source to target language at either the syntactic or semantic level. Simple parsing has been used to detect phrasal boundaries for providing prosodic cues for use in speech synthesis. A great deal of attention is being applied currently to the application of syntactic parsing models for language modeling for automatic speech recognition. There have also been successful applications in information retrieval, where it is useful to know the syntactic form and arguments of phrases in retrieved documents in order to increase retrieval accuracy. Along similar lines, parsing has been used to generate database queries from natural language inputs. Finally, there is a great deal of activity in the cognitive psychology community aimed at explaining human sentence processing by modeling it as a parsing process.

1.6 The future

Of course, most linguists would feel rather constrained by the restriction to context-free grammars, and even more so by regular expressions. Unfortunately, there has been very little work in parsing that goes beyond the context-free grammar paradigm in a significant way. On the one hand, there has been work in parsing formalisms such as *lexical-functional grammar, generalized phrase structure grammar*, and *head-driven phrase structure grammar*, but these all depend on a context-free backbone driving a parser with a slightly generalized notion of category. Even approaches to *dependency grammar* parsing and transformational grammar parsing have wound up relying on conversions in large part to context-free representations. *Tree-adjoining* grammar parsers have been developed that use dynamic programming, indexing subanalyses by left and right external boundaries, as well as internal left and right boundaries for adjunction.

2 Discourse Analysis

The area of discourse analysis is concerned with inferring the intended meanings of utterances with respect to the world in which they were uttered, as

well as with determining how utterances relate to one another to form a coherent structure. Discourse analysis is pertinent both to understanding monologue texts, as well as to correctly interpreting a conversational partner's utterances and subsequently providing an appropriate response in multi-party dialogues. For instance, consider the following exchange between a bank teller (T) and a customer (C):

(1) C: *I want to transfer some money from my checking account to my credit card account.*
(2) T: *What's your checking account number?*
(3) C: *It's 123 456 789.*
(4) T: *How much would you like to transfer?*
(5) C: *I'd like to pay off the balance on my credit card.*
(6) T: *Your credit card balance is $1,036.23, but you only have $962.14 in your checking account.*
(7) C: *Okay.*
(8) C: *I'd like to pay the balance due today then.*
(9) T: *Okay, I have transferred $792.02 from your checking account to your credit account.*

In order for the dialogue participants to successfully carry out a dialogue such as the above, they must be able to recognize the *intentions* of the other participant's utterances, and to produce their responses in such a way that will enable the other participant(s) to recognize their intentions. For instance, although utterance (5) does not directly provide an answer to the question in (4), C produced it with the intention of it being a sufficient answer to T's question based on the assumption that T, being a bank teller, will be able to look up the credit card balance. Furthermore, C must be able to recognize that T's utterance in (6), in addition to informing C of the balances in both accounts, is intended to convey to C the invalidity of the combined proposed action in (1) and (5). Finally, utterance (8) is intended as an alternative plan to satisfy a slight variation of C's original goal.

Discourse processing has been widely studied from a computational point of view since the late 1970s. Up until the last few years, the computational models developed for discourse analysis have, for the most part, taken a plan-based approach based on the speech act theory (Austin 1962, Searle 1975). Within this framework, the speaker is considered to have some goal he or she wishes to achieve, and the utterances in the discourse, whether they be part of a monologue or a dialogue, are actions that the speaker is carrying out in order to achieve the intended goal. Allen and Perrault 1980, and Cohen and Perrault 1979 pioneered this work in plan-based discourse processing in their efforts to formulate a part of Austin's (1962) speech act theory within a computational framework. Their work focussed on formulating simple speech acts such as *request* and *inform* within a plan-based computational framework, and on recognizing indirect speech acts within the simple domain of providing

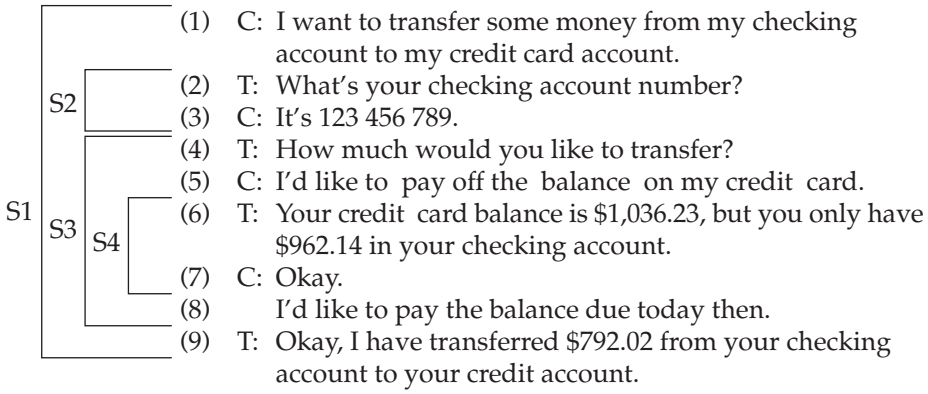

Figure 25.1 Sample dialogue and its discourse segments

users with information about either meeting or boarding a train. Since then, strategies have been developed to handle more complicated tasks and more sophisticated phenomena, such as incremental recognition of plans (Carberry 1990), and the recognition of ill-formed plans (Pollack 1990).

As discussed earlier, the goal of discourse processing is to understand utterances with respect to the context (and the world), and to relate utterances to one another. To illustrate this process, we further analyze the above dialogue segment, which is shown again in figure 25.1 with additional discourse segment information included. In order to understand the overall dialogue segment S1, we must recognize that the dialogue is carried out in order to achieve C's goal of transferring money from a checking account to a credit card account, expressed by C in utterance (1). Furthermore, we must recognize that dialogue segments S2 (comprising utterances (2) and (3)) and S3 (comprising utterances (4)–(8)) are subdialogues initiated by T in order to solicit missing information for carrying out C's intended action, in this case, C's checking account number and the amount to be transferred, for segments S2 and S3, respectively. Finally, we should recognize segment S4 (utterances (6) and (7)) as a subdialogue initiated by C to resolve a detected conflict between C and T with respect to the validity of the plan proposed by C. In this case, T chooses to resolve this conflict by providing evidence of why C's original proposal is invalid (utterance (6)) and then C subsequently proposes a valid alternative to satisfy a slightly different goal (utterance (8)). Based on this analysis, the structure of this dialogue segment may be represented by the tree structure shown in figure 25.2.

A plan-based discourse understanding system may be used to infer the structure of a discourse, such as that shown in figure 25.2, given utterances (1)–(9). The system infers the discourse structure in an incremental fashion, by modeling the current discourse structure and determining how the next utterance can best be incorporated into the current structure to form a coherent

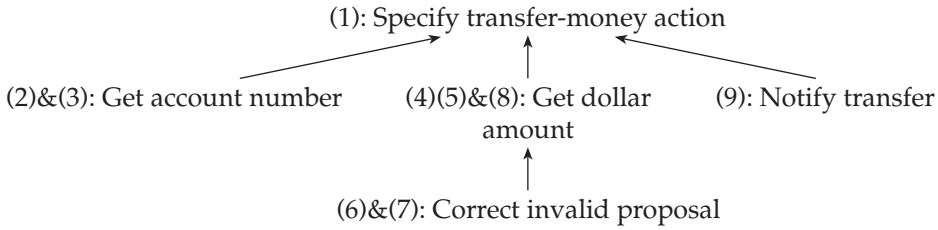

Figure 25.2 Dialogue structure for utterances (1)–(9)

piece of text or dialogue. For instance, given utterances (1)–(7), the discourse understanding system would infer the part of figure 25.2 that includes these utterances, i.e., the bulk of the left and center branches of the tree. Given utterance (8), the system must then determine how this utterance best fits in with the existing discourse structure, i.e., whether utterance (8) contributes to the top level action specified by utterance (1), to the "get dollar amount" action specified by utterances (4) and (5), to be "correct invalid proposal" action specified by utterances (6) and (7), or initiates a new topic that forms the root node of a separate discourse tree. A discourse understanding system capable of performing such a task typically contains three components: (1) a library of generic *recipes* (Pollack 1986), (2) the *plan inference* module, and (3) the system's private knowledge about the domain and about the world, and (optionally) its beliefs about the dialogue participants and their beliefs. Below we sketch the outline of the discourse understanding process.

A recipe is a generic template for performing a particular action. It typically consists of a *header* that specifies the action being described, a set of *preconditions* that specifies the conditions that must hold before the action is executed, the *body* of the action, which comprises the subactions that must be performed as part of performing the header action, and one or more *goals*, which are what the person intended to achieve by performing the action. A recipe for the top-level action in figure 25.2, *Transfer-Money*, is shown in figure 25.3. The header of this recipe shows that the *Transfer-Money* action takes five parameters: *?teller*, who is the agent performing the transfer action, *?customer*, the agent whose money is being transferred, *?from-acct-type*, the type of account from which money will be drawn, *?to-acct-type*, the type of account to which money will be deposited, and *?amount*, the amount of money to be transferred between the specified accounts. The preconditions indicate that before the body of the action can be executed, *?teller* must know both account numbers and that the balance in the "from account" must be greater than the amount to be transferred. The body of the action consists of two subactions, *Transfer*, which is the actual performance of the transfer request, and *Notify-Transfer*, in which *?teller* notifies *?customer* that the transfer request has been completed. Finally, the goal of performing the *Transfer-Money* action is to increase the balance in the "to account" by the amount transferred.

Header:	Transfer-Money (?teller,?customer,?from-acct-type,?to-acct-type,?amount)
/ Gloss: ?teller transfers ?amount from ?customer's ?from-acct-type to ?to-acct-type *|*	
Preconditions:	knowref(?teller,?from-acct-num,account-number(?from-acct-type,?from-acct-num))
	knowref(?teller,?to-acct-num,account-number(?to-acct-type,?to-acct-num))
	greater-than(balance(?from-acct-num),?amount)
Body:	Transfer(?teller,?from-acct-num,?to-acct-num,?amount)
	Notify-Transfer(?teller,?customer,?from-acct-type,?to-acct-type,?amount)
Goal:	increase-balance(?to-acct-type,?amount)

Figure 25.3 Recipe for the "transfer-money" action

The recipe library contains a collection of generic recipes such as the above, and during discourse understanding, the plan inference module attempts to infer utterance intentions and relationships using information provided by this library. The plan inference process begins with the recognized semantic interpretation of the speaker's utterance, which is taken to be the observed action. First, plan inference rules are applied to the observed action to infer other higher level actions that may have resulted in the execution of the observed action. This process, called *forward chaining*, hypothesizes parent actions of observed or inferred actions, and results in a chain of hypothesized actions. A parent action (A_i) may be chained to an inferred or observed action (A_j) if one of two conditions holds. First, A_j may be in the body of the recipe for performing A_i; in other words, performing A_j *contributes* to performing A_i (Pollack 1986). In this case, we may hypothesize that A_j is performed as part of carrying out the higher-level action A_i. Second, the goal of A_j may match a precondition in A_i; in other words, performing A_j *enables* performing A_i (Pollack 1986). In this case, we may hypothesize that A_j is performed in order to make it possible to perform the higher-level action A_i.[1] Next, the plan inference process attempts to incorporate the chain of hypothesized actions (inferred from the new utterance) into existing discourse to form a coherent structure. The same basic inference process is again used to link the root node of the chain to some existing node in the discourse structure so that the root node of the chain either *contributes to* or *enables* an action in the existing discourse structure. When multiple interpretations are plausible, i.e., when the root node of the chain can be linked to more than one existing action, additional heuristics may be used to select from the possible interpretations. One such heuristic is the *focussing rule* (McKeown 1985, Grosz and Sidner 1986, Litman and Allen 1987), which prefers linking the chain to the node that is currently in focus in the discourse structure.

Throughout the discourse understanding process, knowledge about the application domain and about the world comes into play. For instance, knowledge

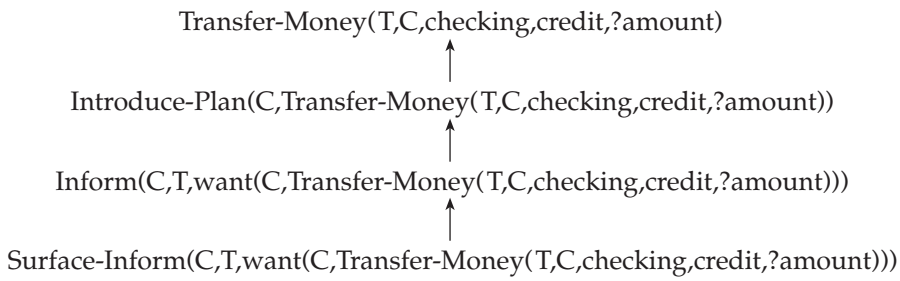

Figure 25.4 Existing discourse model after utterance (1)

```
Header:         Request-ref(?speaker,?hearer,?var,?prop)
/* Gloss: ?speaker asks ?hearer the referent of ?var, which is a parameter in ?prop */
Preconditions:  knowref(?speaker,?var,?prop)
                believe(?speaker,knowref(?hearer,?var,?prop))
Body:           Wh-Question(?speaker,?hearer,?var,?prop)
Goal:           want(?hearer,Answer-Ref(?hearer,?speaker,?var,?prop))

Header:         Obtain-info-ref(?speaker,?hearer,?var,?prop)
/* Gloss: ?speaker obtains from ?hearer the referent of ?var */
Preconditions:  knowref(?speaker,?var,?prop)
                believe(?speaker,knowref(?hearer,?var,?prop))
Body:           Request-Ref(?speaker,?hearer,?var,?prop)
                Answer-Ref(?hearer,?speaker,?var,?prop)
Goal:           knowref(?speaker,?var,?prop)
```

Figure 25.5 Additional recipes for recognizing utterances in figure 25.1

about the domain allows us to recognize that a list of accounts for a user can be easily obtained by a teller, and therefore satisfying one of the first two preconditions in figure 25.3 (obtaining either *?from-acct-num* or *?to-acct-num*) automatically satisfies the other. Furthermore, common sense knowledge allows us to understand what the *greater-than* predicate means in the last precondition and to know that all valid instantiation for the variable *?amount* must be either integers or fixed point numbers with one or two decimal places.

Given this brief overview of the discourse understanding process, we now illustrate how a portion of utterance (1)–(9) can be recognized to form part of the discourse structure in figure 25.2. We show how utterances (2) and (3) are interpreted with respect to utterance (1), which we assume to have already been interpreted and incorporated into the existing discourse model shown in figure 25.4. Utterance (2), based on its semantic representation, is recognized as a *Wh-Question*. Since *Wh-Question* is in the body of the *Request-Ref* action whose recipe is shown in figure 25.5, by forward chaining, the recognition system hypothesizes that utterance (2), intended as a *Wh-Question*, is performed

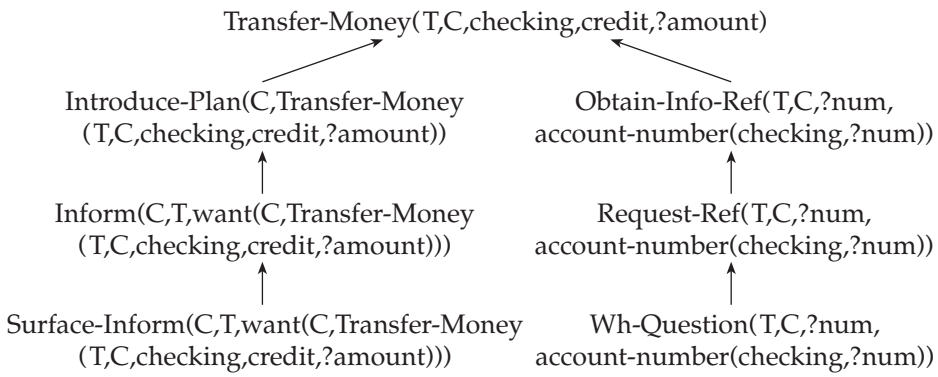

Figure 25.6 Existing discourse model after utterance (2)

in order to perform a *Request-Ref* action. Furthermore, since *Request-Ref* is one of the subactions in the body of *Obtain-Info-Ref* (figure 25.5), forward chaining again leads the recognition system to infer *Obtain-Info-Ref* as the parent action as *Request-Ref*. Since the goal of *Obtain-Info-Ref (?teller,?customer,?num, account-number (checking,?num))* is that *?teller* knows *?customer*'s checking account number, which matches the second precondition for the *Transfer-Money* action, this *Obtain-Info-Ref* action is inferred to have been performed in order to enable the *Transfer-Money* action in the existing discourse structure. Figure 25.6 shows the existing discourse structure after utterance (2).

Utterance (3), on the other hand, is recognized as a *Surface-Inform* action, which is again hypothesized to be an *Inform* action. The recognition component then attempts to incorporate this chain of actions into the existing discourse model by hypothesizing the antecedent actions of *Inform* and determining if each hypothesized chain of actions can be coherently linked to some action in the existing discourse model. One hypothesized parent action of *Inform* is *Answer-Ref*, which is the second subaction of *Obtain-Info-Ref*. As a result, utterance (3) is recognized as providing a response to the question in utterance (2). This analysis leads to the root node and the left branch of the dialogue structure shown in figure 25.2.

A closer analysis of the actions in the discourse model in figure 25.6 shows that our current representation of the discourse model conflates three different types of actions. First, there are *discourse actions* that describe the communicative actions being carried out by each dialogue participant, including actions such as *Inform* and *Request-Ref*. Second, there are *domain actions* (Litman and Allen 1987) that specify the domain-specific actions that the dialogue participants have chosen to satisfy their domain goal, such as *Transfer-Money*. Finally, there are *problem-solving actions* (Allen 1991, Lambert and Carberry 1991, Ramshaw 1991) which are meta-level actions describing how the dialogue participants are going about constructing their domain plan. Examples of problem-solving actions include *Introduce-Plan*, *Evaluate-Plan*, and *Instantiate-Parameter*.

For a decade, researchers have developed models for discourse analysis that distinguish between domain and discourse actions (Litman and Allen 1987), and more recently, models that further distinguish between domain and problem-solving actions (Lambert and Carberry 1991, Ramshaw 1991). By distinguishing among these three types of actions, these models are able to apply action-type-specific heuristics to the recognition process at each level while maintaining the overall uniform plan inference process across all levels. In addition, by distinguishing between domain-independent discourse (and problem-solving) actions and domain-specific actions, the models can be more easily ported to new domains.

Finally, although in this section we have focussed on plan-based discourse analysis, this is by no means the only method employed for computational discourse analysis. It had, however, been the most widely adopted method from the late 1970s until the early 1990s. Recently, with the success of applying statistical methods to other areas of natural language processing, such as part-of-speech tagging and parsing, researchers have started exploring applying such methods to problems in discourse processing, such as discourse segmentation, discourse act recognition, etc. However, such work is still in its infancy, and we expect much progress to be made with respect to statistical discourse analysis in the next few years.

3 Computational Morphology and Phonology

In many areas of computational linguistics it is commonplace to treat words as if they were atomic units with no internal analysis. A syntactic parser, for example, might consider the fact that *eats* is a third singular present verb form; but it would generally be of no interest to a parser that this word can be *morphologically* analyzed into two components, namely a verb stem *eat* and an affix +*s*, which marks form as being third singular present.

There are, nevertheless, applications of natural language technology where such considerations become more important. In text retrieval, for example, the problem is to search a large text database for a term or collection of terms, and to retrieve documents containing those terms. Frequently one is interested not only in finding the exact term, but also in finding morphological variants of that term: a search for *foxes* should retrieve documents containing the word *fox*, for instance.

As a second example, consider text-to-speech synthesis. In many languages, how one pronounces a string of letters depends in large part on what the morphological analysis of that string of letters is. It helps, for example, to know that the analysis of *misled* is *mis+led* in order to avoid pronouncing it as [mízəld] (*misle+d*).

This section will illustrate some basic approaches to word-form analysis. Roughly speaking, the topics that will be covered can be classified into

computational morphology, which treats the analysis of word structure per se; and *computational phonology*, which (to a first approximation at least), deals with the changes in sound patterns that take place when words are put together. Since analysis of word structure invariably presumes that one can "untie" the effects of sound changes, we will start with computational phonology.

3.1 Computational phonology

As a straightforward example of phonological alternations within words, consider the formation of partitive nouns in Finnish. The partitive affix in Finnish has two forms, *-ta*, and *-tä*; the form chosen depends upon the final harmony-inducing vowel of the base. Bases whose final harmony-inducing vowel is back take *-ta*; those whose final harmony-inducing vowel is front take *-tä*. The vowels *i* and *e* are not harmony inducing; they are transparent to harmony. Thus in a stem like *puhelin* "telephone," the last harmony-inducing vowel is *u*, so the form of the partitive affix is *-ta*.

(1)

Nominative	*Partitive*	*Gloss*
taivas	taivas+ta	"sky"
puhelin	puhelin+ta	"telephone"
lakeus	lakeut+ta	"plain"
syy	syy+tä	"reason"
lyhyt	lyhyt+tä	"short"
ystävällinen	ystävällinen+tä	"friendly"

This alternation is part of a much more general *vowel harmony* process in Finnish, one which affects a large number of affixes. Within theoretical linguistics, there have been various approaches to dealing with phonological alternations such as the one exemplified in (1). For the purposes of this discussion we can assume a traditional string-based rewrite-rule analysis, rather than a more up-to-date prosodic declarative analysis: from the point of view of computational phonology, our choice here is largely irrelevant, since it is roughly equally straightforward to implement a modern analysis as it is a more traditional analysis. The latter is a little easier to understand, however.

A rewrite analysis – one that is an oversimplification of Finnish vowel harmony, but one that will do for current purposes – is given below:

(2) a → ä / [ä, ö, y] C* ([i, e] C*)* ___

This rule simply states that an *a* is changed into *ä* when preceded by a vowel from the set *ä, ö, y*, with possible intervening *i, e* and consonants. This rule thus makes a particular assumption about the alternation in (2), namely that in a form like *-tä*, the vowel is underlyingly /a/, and that the underlying form of

the partitive affix is therefore *-ta*, a form that surfaces only when preceded by back and neutral vowels.

A computational model that implements this alternation would seek to change /a/ into /ä/ in the appropriate environment; this is of course stated from the point of view of generating a surface form from an underlying sequence of morphemes. An equally legitimate (and actually much more widespread) interpretation is that one wishes to reconstruct the lexical form from the surface form: thus from *syytä* one wishes to reconstruct *syy+ta*. An ideal computational model would make these two interpretations, *generation* and *analysis*, equally easy to implement. One important property of *finite-state transducers* – the most widespread computational device that is used to implement phonological rules – is precisely this *reversibility*.

A finite-state transducer can be thought of as a variant on a *finite-state acceptor* (also often termed *finite-state automaton*), something familiar to anyone who has taken an introductory formal language theory course (see, e.g., Hopcroft and Ullman 1979). A finite-state acceptor, it will be recalled, is a device that has a finite number of *states* (hence its name), one of which (typically) is designated as a *start* state, and some subset of which are designated as *final* states. Transitions between states are termed *arcs* – there may be zero or more arcs going from any state s_i to any other state s_j – and each arc is *labeled* with a single element from a predesignated *alphabet*, or else it is labeled with the *empty string* element ε, in which case it is termed a *null arc*. A given *string* made up of elements from the alphabet is accepted by a given acceptor if and only if one can start at the start state of the acceptor, move from one state to the next matching the label of the arc against the next element of the input, and end up in a final state, having consumed *all* of the input. The set of strings accepted by an automaton is termed the *language* of that automaton, and the set of all languages accepted by all possible finite-state automata is the set of *regular languages*.

A finite-state transducer differs from a finite-state acceptor in having, not a single label on an arc, but rather a pair of labels, one being the *input label*, the other the *output label*. The machine works much as does an automaton, except that in addition to matching the input label against the symbol of the input, one also replaces it with the output label. Thus transducers transduce input strings into potentially different output strings. Note that either the input label or the output label might be ε: in the former case, one may transition an arc matching nothing on the input side (and thus consuming no input), but at the same time *inserting* a symbol on the output side; in the latter case, one must match the input label against the input, but the effect of transiting the arc is to *delete* the input symbol. Since transducers relate sets of pairs of (input and output) strings, they are said to compute *relations*: the set of relations computed by all possible finite-state transducers is termed the set of *regular relations*.

Returning now to phonology, it has been known for several decades (Johnson 1972, Kaplan and Kay 1994), that systems of standard phonological rewrite

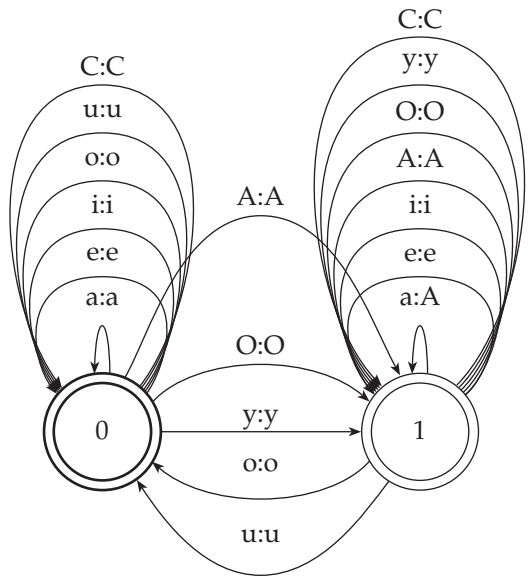

Figure 25.7 An FST implementation of the rule in (2)
Note: *A* represents *ä*, *O* represents *ö*, and *C* represents any consonant. The labels on the arcs represent the symbol-to-symbol transductions, with the input on the left and the output on the right of the colon.

rules, given certain restrictions, constitute regular relations: that is, the relation between the set of all possible lexical forms of a language, and the set of all possible surface forms is a regular relation. This means that phonological rule systems can be modeled as finite-state transducers, and there has to date been a large amount of work that develops this idea (e.g., Koskenniemi 1983, Karttunen 1983, Karttunen et al. 1992), and explicit methods for *compiling* transducers from phonological rewrite rule descriptions have been developed (e.g., Kaplan and Kay 1994, Mohri and Sproat 1996). The desired *reversibility*, discussed above, comes about quite simply. Suppose one has constructed a transducer that maps from lexical forms (as input) to surface forms (as output). One can produce a device that maps the other way by simply inverting the input and output labels on each arc.

It is time to give an example of a finite-state transducer, and for this we return to the alternation given in (1) and described by the rule in (2). A transducer that implements this rule is shown in figure 25.7. In this transducer, state 0 is the initial state, and both states 0 and 1 are final states. The machine stays in state 0, transducing *a* to itself, until it hits one of the front vowels in the input *ä*, *ö*, *y*, in which case it goes to state 1, where it will transduce input *a* to *ä*.

The approach to computational phonology outlined here is fairly traditional, and it presumes a model of phonology that is certainly not current. More

up-to-date computational models include *declarative* approaches such as (Bird and Ellison 1994), as well as various implementations of *optimality theory*. In many cases, the computational devices used are formally rather similar to the transducers discussed here; in Bird and Ellison's case, for instance, finite-state acceptors are used. This is an interesting point to take note of, since while much has recently been made of the advantages of declarative, constraint-based approaches over traditional rule-based approaches, at the computational level, at least, the differences between models are often rather minimal.

3.2 *Computational morphology*

As with computational phonology, the most popular computational models of word structure have been finite-state ones (or ones that are mathematically equivalent to finite-state). This is by no means the only approach that has been taken, and indeed there are certainly limitations in finite-state methods, as has been discussed elsewhere (e.g., Sproat 1992). Nonetheless, a sufficiently rich array of morphological phenomena can be handled with finite-state devices that it will be reasonable to restrict our attention to finite-state approaches here.

As a concrete example of computational morphological modeling, let us consider a simple subset of Spanish verbal morphology, consisting of three verbs, *hablar* "speak," *cerrar* "close," and *cocer* "cook," conjugated in the present and preterite indicative forms; these are given in table 25.1. Note that these

Table 25.1 Some regular verbal forms in Spanish

Features	*hablar*	*cerrar*	*cocer*
ind pres 1sg	hablo	c**ie**rro	c**uez**o
ind pres 2sg	hablas	c**ie**rras	c**ue**ces
ind pres 3sg	habla	c**ie**rra	c**ue**ce
ind pres 1pl	hablamos	cerramos	cocemos
ind pres 2pl	habláis	cerráis	cocéis
ind pres 3pl	hablan	c**ie**rran	c**ue**cen
ind pret 1sg	hablé	cerré	cocí
ind pret 2sg	hablaste	cerraste	cociste
ind pret 3sg	habló	cerró	coció
ind pret 1pl	hablamos	cerramos	cocimos
ind pret 2pl	hablasteis	cerrasteis	cocisteis
ind pret 3pl	hablaron	cerraron	cocieron

Table 25.2 An arclist model of Spanish verbal morphology

START	ar	habl	hablar **vb**
START	ar	cerr **diph**	cerrar **vb**
START	er	coc **diph c/z**	cocer **vb**
ar	WORD	+o#	+**ind pres 1sg**
ar	WORD	+as#	+**ind pres 2sg**
ar	WORD	+a#	+**ind pres 3sg**
ar	WORD	+amos#	+**ind pres 1pl**
ar	WORD	+'ais#	+**ind pres 2pl**
ar	WORD	+an#	+**ind pres 3pl**
ar	WORD	+'e#	+**ind pret 1sg**
ar	WORD	+aste#	+**ind pret 2sg**
ar	WORD	+'o#	+**ind pret 3sg**
ar	WORD	+amos#	+**ind pret 1pl**
ar	WORD	+asteis#	+**ind pret 2pl**
ar	WORD	+aron#	+**ind pret 3pl**
er	WORD	+o#	+**ind pres 1sg**
er	WORD	+es#	+**ind pres 2sg**
er	WORD	+e#	+**ind pres 3sg**
er	WORD	+emos#	+**ind pres 1pl**
er	WORD	+'eis#	+**ind pres 2pl**
er	WORD	+en#	+**ind pres 3pl**
er	WORD	+'i#	+**ind pret 1sg**
er	WORD	+iste#	+**ind pret 2sg**
er	WORD	+i'o#	+**ind pret 3sg**
er	WORD	+imos#	+**ind pret 1pl**
er	WORD	+isteis#	+**ind pret 2pl**
er	WORD	+ieron#	+**ind pret 3pl**
WORD			

three verbs come from two conjugations, namely the *-ar* conjugation (*hablar* and *cerrar*) and the *-er* conjugation (*cocer*). Also note the "spelling changes" in some of the stem forms, in particular the diphthongization of the stem vowel in certain positions in *cerrar* (*cierr-*) and *cocer* (*cuec-*), and the *c*/*z* alternation in the stem of *cocer*.

A simple finite-state model that allows one to recognize these verb forms is what we will term an *arclist* model, following (Tzoukermann and Liberman 1990), represented in table 25.2. Verb stems are represented as beginning in the initial state, and going to a state which records their paradigm affiliation, *-ar* or *-er*. From the paradigm states, one can get to the final WORD state by means of appropriate endings for that paradigm. The verb stems for *cerrar* and *cocer*

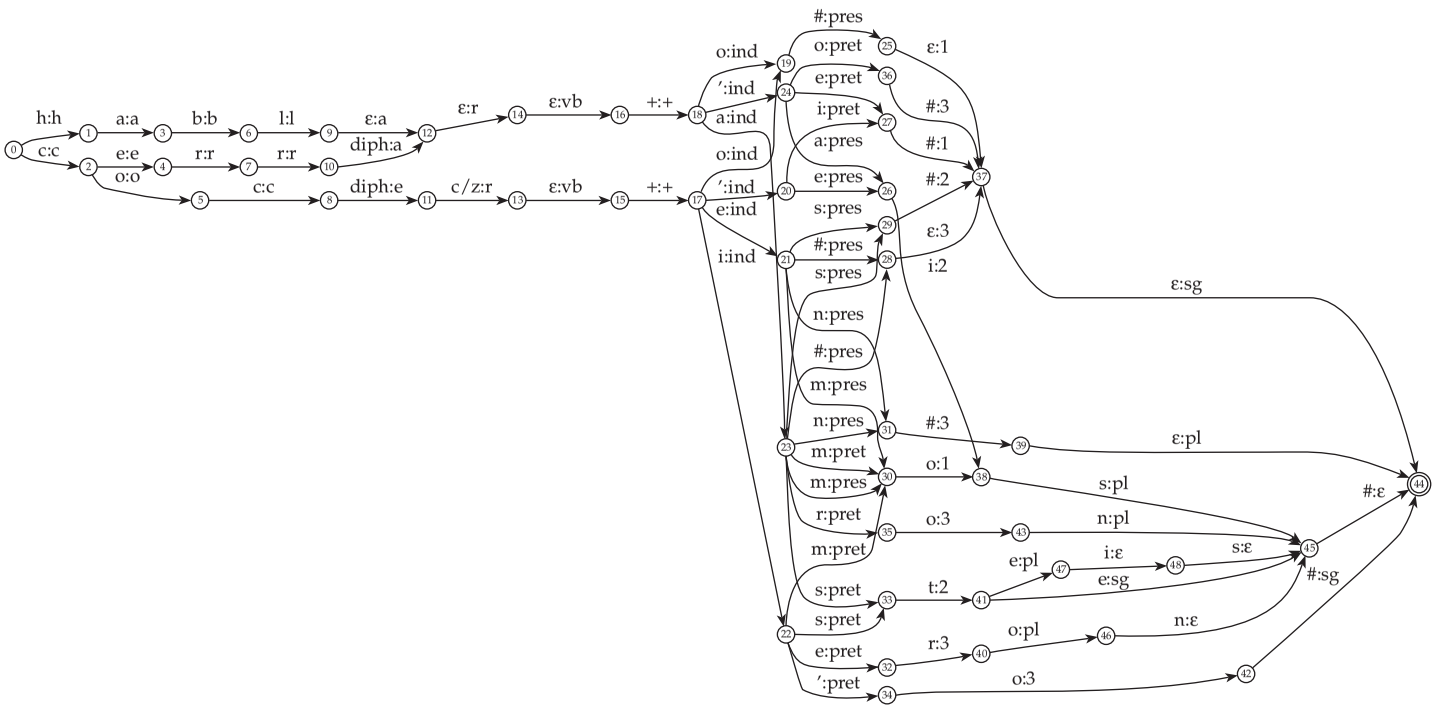

Figure 25.8 A transducer for a small fragment of Spanish verbal morphology

have lexical features **diph** and **c/z**, which trigger the application of spelling changes. This arclist can easily be represented as a finite-state transducer; see figure 25.8.

The spelling changes necessary for this fragment involve rules to diphthongize vowels, change *c* into *z*, and delete grammatical features and the morpheme boundary symbol. Thus the first of the following set of rules changes *e* into *ie* if it is followed by zero or more consonants (C*), zero or more lexical features (**feat***), the lexical feature **diph**, possible other features, and then a morpheme boundary followed by a vowel and an optional consonant sequence. The second rule similarly changes *o* into *ue*. The third rule changes *c* into *z* in stems marked with the **c/z** feature before [+back] vowels. Finally the last rule deletes lexical features and boundary symbols:

(3) e → ie / __ C* **feat*** **diph** **feat*** + V C* #
o → ue / __ C* **feat*** **diph** **feat*** + V C* #
c → z / __ **feat*** **c/z** + [+back]
(**feature** ∨ **boundary**) → ε

(Note that it is *not* claimed that this is a correct set of rules for handling Spanish morphographemics in general: it is merely presented as a solution for the small fragment under discussion.) Following the discussion in section 3.1, we can represent this set of rules as a finite state transducer. One useful property of finite state transducers is that they are *closed under composition*: that is, if one has a finite state transducer T_1, that maps from set x to set y, and another transducer T_2 that maps from set y to set z, one can compose T_1 and T_2 together – notated as $T_1 \circ T_2$ – to obtain a third transducer T_3, that maps from x to z. Armed with this, we can produce a transducer that maps directly from lexical to surface forms, by simply composing the transducer representing the lexicon (figure 25.8) with the transducer representing the rules in (3). As we also noted in section 3.1, finite state transducers are invertible; hence one can simply invert this transducer to obtain one that maps from surface to lexical forms. This transducer is represented in figure 25.9.

Strictly finite-state models of morphology are in general only minor variants of the model just presented. For example, the original system of Koskenniemi (Koskenniemi 1983) modeled the lexicon as a set of letter tries, which can be thought of as a special kind of finite automaton. Morphological complexity was handled by *continuation patterns*, which were annotations at the end of morpheme entries indicating which set of tries to continue the search in. But, again, this mechanism merely simulates an ε-labeled arc in a finite-state machine. Spelling changes were handled by two-level rules implemented as parallel (virtually, if not actually, intersected) finite-state transducers. Search on the input string involved matching the input string with the input side of the two-level transducers, while simultaneously matching the lexical tries with the lexical side of the transducers, a process formally equivalent to the model we have presented here.

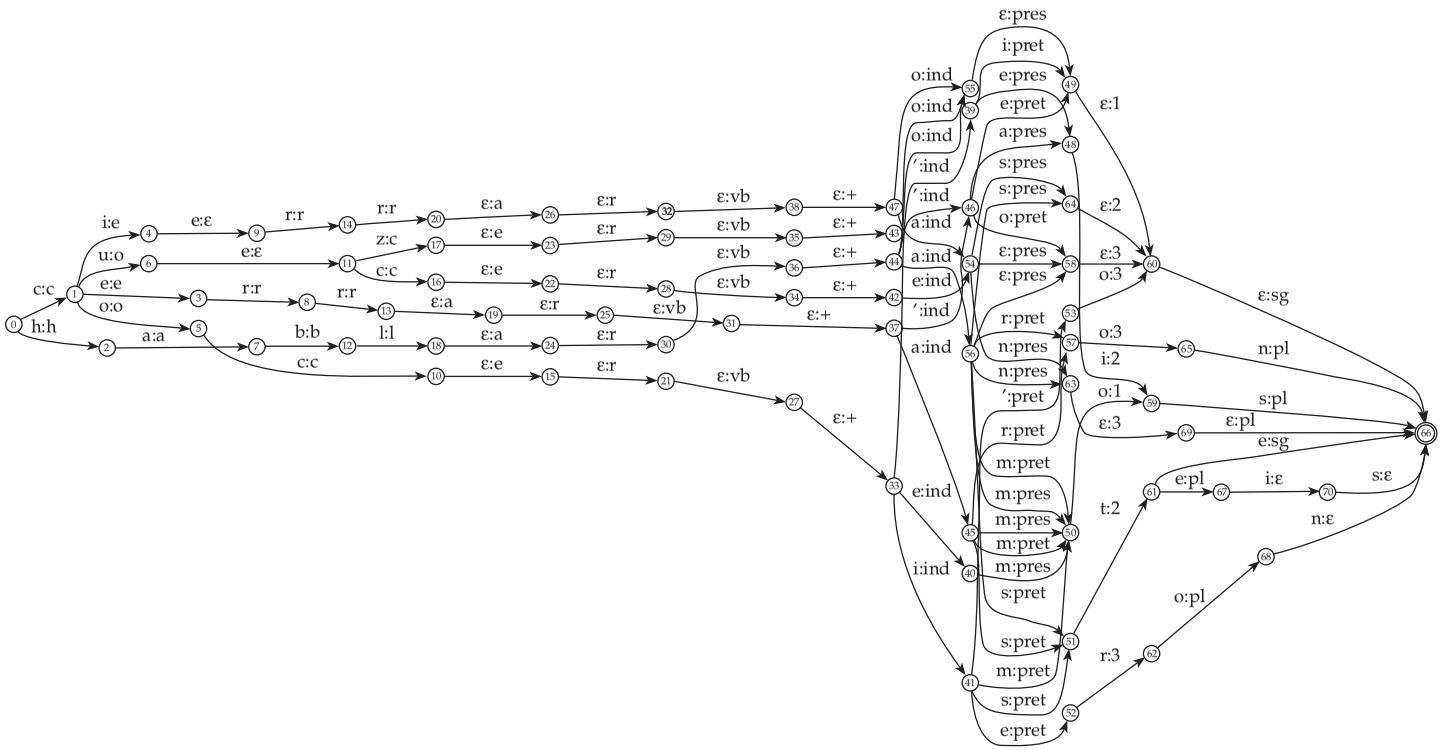

Figure 25.9 A transducer that maps between surface and lexical forms for a small fragment of Spanish verbal morphology

4 Corpus-based Methods

The word *corpus* is borrowed from Latin, in which it means "body." The word corpus is commonly used to refer to a body of writings of some sort, and in linguistics, this is typically a collection of texts. In the speech sciences, a collection of audio recordings, possibly transcribed and labeled, is also often referred to as a corpus. The corpus can consist simply of plain text, or it can be annotated using some set of special symbols. For example, it can be marked up according to an SGML scheme to convey information about the textual structure, which is to be interpreted by, say, a web browser. Or the corpus can be annotated to indicate structure according to some theory of linguistics. One example is assigning a part-of-speech (PoS) tag – a linguistically motivated label – to each word in the text; another example is assigning a parse tree – a hierarchical analysis using some grammar of rewrite rules – to each sentence in the text.

Corpora have been widely used by linguists to identify and analyze language phenomena, and to verify or refute claims about language. However, a corpus is normally considered to be more than the mere sum – or rather concatenation – of its parts, especially in the field of computational linguistics; it also reveals important quantitative information about the distribution of various language phenomena. This ranges from rather trivial observations such as "the most frequent word in almost any English text is the word *the*," to more sophisticated inferences in the vein of "there is a tendency to move long noun phrases towards the end of a clause."

The aspect of conveying quantitative information means that the corpus must in some sense be representative of the (sub)language from which it is drawn. Thus, a collection of "laboratory sentences," such as *That that Kim snores annoyed Sandy surprised Chris*, where each sentence has been included because it exhibits some interesting language phenomenon, does not quite qualify as a corpus. Such a collection of linguistically interesting examples is of course a very useful resource in itself for testing theories or developing language-processing systems, but it is usually referred to as a *test suite*, rather than a corpus. Indeed, much of the work in corpus-based computational linguistics has been concerned with extracting the quantitative information contained in a corpus and reformulating it in a more concise way, i.e., to extract a quantitative *language model* from the corpus. The purpose of the extracted language model is usually to study language as such, or to build language-processing devices. There are, however, other uses for such a compact description of language: speech recognition and text compression.

In text compression, the goal is to save storage space, and to this end, a language model can be very useful. And indeed, very similar work has progressed in parallel in the fields of language modeling and text compression. The scenario here is that you wish to encode a text as a sequence of binary digits. We can of course simply write down the seven-bit ASCII representation of each character in the text, and we are done. We could alternatively find

some numbering of the words and instead write down the binary representation of the word number, most likely saving a lot of storage space. We could save even more space by having different lengths of the binary representations of the words, and encode more frequent words using shorter digit sequences. This is the basic idea behind coding schemes such as *Huffman coding* (see, e.g., Crochemore and Rytter 1994).

In speech recognition, we wish to try to predict the next word of an utterance, given the previous sequence of words in it, to constrain the search space of the speech recognizer. Also to this end, a language model can be very useful, and indeed, speech recognition has been a driving force in language modeling research. In particular, the word bigram model, to be discussed shortly, has proved crucial for much of the progress in the field of speech recognition.

Let us consider a very simple language model, namely a word-pair grammar. This grammar specifies what word can follow what other word. It might for example allow the word *computer*, but not the word *is*, to follow the word *the*. Such a model can easily be extracted from a plain-text corpus by simply observing what words follow each other in the corpus. This language model is best represented as a finite-state automaton (FSA). Each state of the automaton corresponds to a word, and a directed arc from one state to another indicates that the word of the second state can follow the word of the first state.

This model does not, however, capture any quantitative information. To remedy this, we add probabilities to the scheme; we now specify the probability of any word following any other word. For example, we might stipulate that the probability of the next word being *computer*, given that the current one is *the*, is 0.0042, while the probability of the next word being *is* is exactly zero. This is known as a *word bigram model*, and the probabilities are called *bigram probabilities*. This model is best represented as a probabilistic finite-state automaton. We now attribute a transition probability to each arc, which is the conditional probability of the next state given the current one, which in turn is just the bigram probability. We could in fact use this model to generate sentences: given the current state, i.e., the current word, we draw a next state at random, according to the transition probabilities, transit to it, and output the word corresponding to the state reached, etc. Such a model is known as a *generative language model*, which is the kind of language model used in, e.g., speech recognition.

The bigram probabilities can be estimated from a plain-text corpus by simply counting relative frequencies: if the word *the* occurred 10,000 times in the corpus, and the word *computer* followed it 42 times, we estimate the corresponding bigram probability to be $\frac{42}{10{,}000} = 0.0042$. If, on the other hand, we saw no instance of the word *is* following the word *the*, we assign this bigram probability the value zero. However, a little afterthought yields us the insight that these zero probability bigrams might not be all that desirable; just because we didn't see the word *small* following the word *the* doesn't mean that it cannot do so in new, hitherto unseen texts, which the zero probability would imply.

To remedy this, we resort to the black art of *smoothing* the probability estimates. This typically takes the form of a Robin Hood strategy, stealing probability mass from those who have, and giving it to those who have not. As a concrete example, to avoid the zero bigram probabilities, let us simply assign half a count to any word with zero observed count when doing the relative-frequency estimate. This means that the probability of the word *small* following the word *the* will be $\frac{\frac{1}{2}}{15{,}000} = 0.000033$.[2] Indeed, much research in language modeling for speech recognition has been on the theme of improving the smoothing techniques.

Let us now turn to linguistically annotated corpora. The perhaps simplest, and currently most popular approach is to assign to each word in the corpus a *part-of-speech (PoS) tag*, which is a linguistically motivated label. The set of possible PoS tags can consist of a small set of atomic labels – such as the basic word classes adjectives, adverbs, articles, conjunctions, nouns, numbers, prepositions, pronouns, and verbs, essentially introduced already by the ancient Greek Dionysius Thrax – or of a complex hierarchical tag set[3] such as the one used in the annotation of the Susanne corpus (Sampson 1995), which consists of over 400 different tags. The tags can even indicate the syntactic role of each word, such as subject, object, adverbial, etc.

Such a representation can be used for disambiguation, as in the case of the well-known, highly ambiguous, and very popular example sentence *Time flies like an arrow*. We can for example prescribe that *Time* is a noun, *flies* is a verb, *like* is a preposition (or adverb, according to taste), *an* is an article, and that *arrow* is a noun. We realize that words may be assigned different labels in different contexts, or in different readings; for example, if we instead prescribe that *flies* is a noun and *like* is a verb, we get another reading of the sentence. Thus, achieving the most appropriate reading of any given sentence can be done by finding the most likely assignment of tags to the words in it. There is an appropriate extension to the probabilistic finite-state automata encoding the word bigram models that is well suited for this task, namely *hidden Markov models* (*HMMs*). This allows us to automatically make a preferred assignment of PoS tags to previously unseen text. Such a processing tool is known as an *HMM-based PoS tagger*.

To visualize a bigram tagger, consider the FSA representing the word bigram model, but replace the words with PoS tags. This gives us a tag bigram model, where the states correspond to tags. In the word bigram model, we could generate a sentence by moving around between states and outputting the word corresponding to the current state after each move. We will still move around between the states, but instead of outputting the corresponding tag, we will output a word. The word will be drawn at random according to a probability distribution associated with the current state, i.e., with the current tag. So if the current state corresponds to the tag noun, we expect the probability of the word *the* or the word *is* to be very low, if not zero, whereas when we transit to the state corresponding to the tag article, we expect the probability of the word

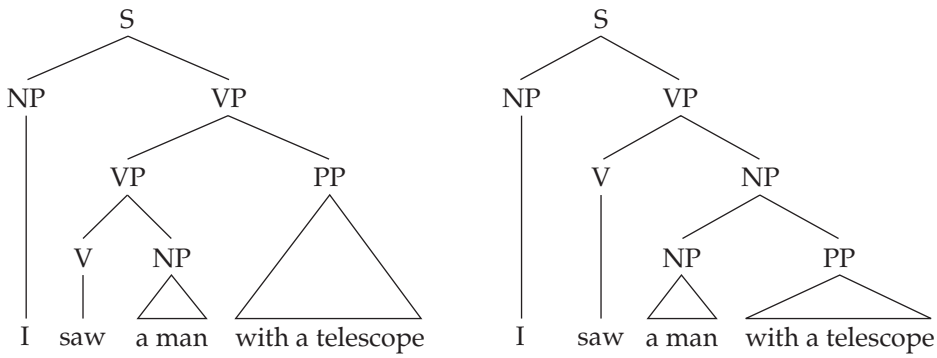

Figure 25.10 Two analyses of "I saw a man with a telescope"

the to rocket. The model is hidden in the sense that we cannot observe the state (tag) sequence, only the word sequence. The model has the *Markov property* since it is modeled by a probabilistic FSA, and the current state is the only memory of the previous history left to condition the transition and word probabilities on. The disambiguation task of finding the most likely tag assignment to the words of a given sentence, i.e., to guess the state sequence that generated the given word sequence, can be performed efficiently by a simple dynamic programming technique known as *Viterbi search*.

Some ambiguity is caused by the way the words relate to each other, rather by what PoS tag is assigned to them. In the case of the equally famous example sentence *I saw a man with a telescope*, the open question is: Who had the telescope, the man or I? We realize that in both cases, we have the same assignment of PoS tags; *I* is a pronoun (Pron), *saw* is a verb (V), *a* is an article (Art) twice, *with* is a preposition (Prep), and *man* and *telescope* are both nouns (N). The way linguists traditionally analyze this is to say that the prepositional phrase (PP) *with a telescope* modifies the word *saw* if I used the telescope to see the man, and modifies the word *man* if the man had the telescope. One way to distinguish between these two alternatives is to assign a parse tree to the sentence. The two parse trees corresponding to the two different analyses are shown in figure 25.10.

The idea here is that some *formal grammar* consisting of *rewrite rules* allows us to rewrite the top symbol[4] as a sequence of strings and the end result is the given sentence. The following simple grammar allows deriving the two different readings of *I saw a man with a telescope* from *S*:

$S \rightarrow NP\ VP$	$Pron \rightarrow I$
$VP \rightarrow V\ NP$	$V \rightarrow saw$
$VP \rightarrow VP\ PP$	$Art \rightarrow a$
$PP \rightarrow Prep\ NP$	$Prep \rightarrow with$
$NP \rightarrow NP\ PP$	$N \rightarrow man$
$NP \rightarrow Pron$	$N \rightarrow telescope$
$NP \rightarrow Art\ N$	

Here is the derivation corresponding to the man having the telescope:

> $S \Rightarrow NP\ VP \Rightarrow Pron\ VP \Rightarrow I\ VP \Rightarrow I\ V\ NP \Rightarrow I\ saw\ NP \Rightarrow I\ saw\ NP\ PP \Rightarrow I\ saw\ Art\ N\ PP \Rightarrow I\ saw\ a\ N\ PP \Rightarrow I\ saw\ a\ man\ PP \Rightarrow I\ saw\ a\ man\ Prep\ NP \Rightarrow I\ saw\ a\ man\ with\ NP \Rightarrow I\ saw\ a\ man\ with\ Art\ N \Rightarrow I\ saw\ a\ man\ with\ a\ N \Rightarrow I\ saw\ a\ man\ with\ a\ telescope$

Since any "grammatical" sentence can be derived from the top symbol *S*, this is also a generative language model.

A *parse tree* is a hierarchical representation[5] of a derivation. You may have noticed that in each derivation step, the leftmost possible grammar symbol was always rewritten. This is called a *leftmost derivation*, and is used to establish a one-to-one correspondence between derivations and parse trees; parse trees don't care in which order the various grammar symbols are rewritten, but derivations do. We realize that we can assign a parse tree to each sentence-level unit in our corpus to specify a particular reading of each one. Such an annotated corpus is usually referred to as a *tree bank* (e.g., Marcus et al. 1993).

We will finally look at a language model that we can extract from a tree bank that allows us to automatically assign preferred parse trees to previously unseen input text. We note that the only difference between the two parse trees, and between the two derivations, of the example sentence is that in one of them, we used the grammar rule $VP \rightarrow VP\ PP$, whereas in the other one, we used the grammar rule $NP \rightarrow NP\ PP$ to attach the prepositional phrase. Now, if we assign probabilities to every grammar rule, we can assign a probability to each parse tree, or derivation, by simply multiplying the probabilities of the rules used. This would allow us to select the parse tree with the highest probability when disambiguating. We recall that in a derivation, we must always rewrite the leftmost possible grammar symbol. This means that we really don't have any choice as to which grammar symbol to rewrite, and it makes sense to have the probabilities sum to one for each left-hand-side (LHS) grammar symbol separately. Such a grammar is called a *stochastic* (or *probabilistic*) *context-free grammar* (*SCFG* or *PCFG*).

The following is our previous grammar, now equipped with probabilities:

$S \rightarrow NP\ VP$	1.00	$Pron \rightarrow I$	1.00
$VP \rightarrow V\ NP$	0.65	$V \rightarrow saw$	1.00
$VP \rightarrow VP\ PP$	0.35	$Art \rightarrow a$	1.00
$PP \rightarrow Prep\ NP$	1.00	$Prep \rightarrow with$	1.00
$NP \rightarrow NP\ PP$	0.25	$N \rightarrow man$	0.58
$NP \rightarrow Pron$	0.30	$N \rightarrow telescope$	0.42
$NP \rightarrow Art\ N$	0.45		

This stochastic grammar tells us to prefer the reading where I had the telescope, since the probability of the rule $VP \rightarrow VP\ PP$, 0.35, is higher than the probability of rule $NP \rightarrow NP\ PP$, 0.25, and since the derivations differ only in the use of these two rules. In practice, we would estimate the rule probabilities

from the relative frequencies of the rules in a tree bank, adding a measure of smoothing techniques for increased robustness.

We conclude by noting that the independence assumptions underlying the statistical model, namely that the only conditioning of the probabilities is on the LHS symbol of the grammar rule, exactly mirror the context-free assumption of the underlying context-free grammar. This means, amongst other things, that for any algorithm for parsing with a context-free grammar, there exists an equally efficient variant of it for finding the most probable parse tree of any given sentence, and for calculating the sentence probability, which is formally defined as the sum of all possible derivation probabilities, under a stochastic context-free grammar.

NOTES

1 In hypothesizing parent actions, in addition to satisfying either the *contribution* or *enablement* relationships, further constraints are placed by examining the preconditions in the recipes for the hypothesized parent actions. We will not provide the details in this chapter; interested readers should refer to Carberry 1990.

2 The denominator increased from 10,000 to 15,000, say, due to all the extra half counts, which in turn reduced the probability of the word *computer* from its previous value 0.0042 to $\frac{42}{15{,}000} = 0.0028$ – Robin Hood at work!

3 A hierarchical tag set is a set of labels with successively finer distinctions, e.g., N for noun, NS for singular noun, NSM for singular masculine noun, NSMN for singular masculine nominative noun, etc.

4 Typically *S*, for sentence.

5 Not to be confused with a hierarchical tag set.

FURTHER READING

A general introduction to natural language understanding, which includes a discussion of parsing and discourse analysis, is Allen 1995. An introductory text to computational morphology and phonology is Sproat 1992. A standard text for introductory formal language theory – a prerequisite to further study of several topics introduced in this chapter – is Hopcroft and Ullman 1979.

There are several papers that address issues in parsing of various particular syntactic theories: for LFG, see Kaplan and Bresnan 1982; for HPSG, Carpenter 1992; for GB, Stabler 1992. Finally for a discussion of some issues related to computational complexity and natural language, see Barton et al. 1987.