In Harry C. Bunt and Anton Nijholt (eds.), *Advances in Probabilistic and Other Parsing Technologies*, Chapter 3, pp. 29-62.

Chapter 1

BILEXICAL GRAMMARS AND THEIR CUBIC-TIME PARSING ALGORITHMS

Jason Eisner
Dept. of Computer Science, University of Rochester
P.O. Box 270226
Rochester, NY 14627-0226 U.S.A.[*]
jason@cs.rochester.edu

Abstract This chapter introduces weighted bilexical grammars, a formalism in which individual lexical items, such as verbs and their arguments, can have idiosyncratic selectional influences on each other. Such 'bilexicalism' has been a theme of much current work in parsing. The new formalism can be used to describe bilexical approaches to both dependency and phrase-structure grammars, and a slight modification yields link grammars. Its scoring approach is compatible with a wide variety of probability models.

The obvious parsing algorithm for bilexical grammars (used by most previous authors) takes time $O(n^5)$. A more efficient $O(n^3)$ method is exhibited. The new algorithm has been implemented and used in a large parsing experiment (Eisner, 1996b). We also give a useful extension to the case where the parser must undo a stochastic transduction that has altered the input.

1. INTRODUCTION

1.1 THE BILEXICAL IDEA

Lexicalized Grammars. Computational linguistics has a long tradition of *lexicalized* grammars, in which each grammatical rule is specialized for some individual word. The earliest lexicalized rules were word-specific subcategorization frames. It is now common to find fully lexicalized versions of many grammatical formalisms, such as context-free and tree-adjoining grammars (Schabes et al., 1988). Other formalisms, such as dependency grammar (Mel'čuk, 1988) and

[*]This material is based on work supported by an NSF Graduate Research Fellowship and ARPA Grant N6600194-C-6043 'Human Language Technology' to the University of Pennsylvania.

head-driven phrase-structure grammar (Pollard and Sag, 1994), are explicitly lexical from the start.

Lexicalized grammars have two well-known advantages. When syntactic acceptability is sensitive to the quirks of individual words, lexicalized rules are necessary for linguistic description. Lexicalized rules are also computationally cheap for parsing written text: a parser may ignore those rules that do not mention any input words.

Probabilities and the New Bilexicalism. More recently, a third advantage of lexicalized grammars has emerged. Even when syntactic *acceptability* is not sensitive to the particular words chosen, syntactic *distribution* may be (Resnik, 1993). Certain words may be able but highly unlikely to modify certain other words. Of course, only some such collocational facts are genuinely lexical (*the storm gathered/*convened*); others are presumably a weak reflex of semantics or world knowledge (*solve puzzles/??goats*). But both kinds can be captured by a *probabilistic* lexicalized grammar, where they may be used to resolve ambiguity in favor of the most probable analysis, and also to speed parsing by avoiding ('pruning') unlikely search paths. Accuracy and efficiency can therefore both benefit.

Work along these lines includes (Charniak, 1995; Collins, 1996; Eisner, 1996a; Charniak, 1997; Collins, 1997; Goodman, 1997), who reported state-of-the-art parsing accuracy. Related models are proposed without evaluation in (Lafferty et al., 1992; Alshawi, 1996).

This flurry of probabilistic lexicalized parsers has focused on what one might call **bilexical grammars**, in which each grammatical rule is specialized for not one but *two* individual words.[1] The central insight is that specific words subcategorize to some degree for other specific words: *tax* is a good object for the verb *raise*. These parsers accordingly estimate, for example, the probability that word w is modified by (a phrase headed by) word v, for each pair of words w, v in the vocabulary.

1.2 AVOIDING THE COST OF BILEXICALISM

Past Work. At first blush, bilexical grammars (whether probabilistic or not) appear to carry a substantial computational penalty. We will see that parsers derived directly from CKY or Earley's algorithm take time $O(n^3 \min(n, |V|)^2)$ for a sentence of length n and a vocabulary of $|V|$ terminal symbols. In practice $n \ll |V|$, so this amounts to $O(n^5)$. Such algorithms implicitly or explicitly regard the grammar as a context-free grammar in which a noun phrase headed by *tiger* bears the special nonterminal NP_{tiger}. These $O(n^5)$ algorithms are used by (Charniak, 1995; Alshawi, 1996; Charniak, 1997; Collins, 1996; Collins, 1997) and subsequent authors.

Speeding Things Up. The present chapter formalizes a particular notion of bilexical grammars, and shows that a length-n sentence can be parsed in time only $O(n^3g^3t)$, where g and t are bounded by the grammar and are typically small. (g is the maximum number of senses per input word, while t measures the degree of interdependence that the grammar allows *among* the several lexical modifiers of a word.) The new algorithm also reduces space requirements to $O(n^2g^2t)$, from the cubic space required by CKY-style approaches to bilexical grammar. The parsing algorithm finds the highest-scoring analysis or analyses generated by the grammar, under a probabilistic or other measure.

The new $O(n^3)$-time algorithm has been implemented, and was used in the experimental work of (Eisner, 1996b; Eisner, 1996a), which compared various bilexical probability models. The algorithm also applies to the Treebank Grammars of (Charniak, 1995). Furthermore, it applies to the head-automaton grammars (HAGs) of (Alshawi, 1996) and the phrase-structure models of (Collins, 1996; Collins, 1997), allowing $O(n^3)$-time rather than $O(n^5)$-time parsing, granted the (linguistically sensible) restrictions that the number of distinct X-bar levels is bounded and that left and right adjuncts are independent of each other.

1.3 ORGANIZATION OF THE CHAPTER

This chapter is organized as follows:

First we will develop the ideas discussed above. §2. presents a simple formalization of bilexical grammar, and then §3. explains why the naive recognition algorithm is $O(n^5)$ and how to reduce it to $O(n^3)$.

Next, §4. offers some extensions to the basic formalism. §4.1 extends it to weighted (probabilistic) grammars, and shows how to find the best parse of the input. §4.2 explains how to handle and disambiguate polysemous words. §4.3 shows how to exclude or penalize string-local configurations. §4.4 handles the more general case where the input is an arbitrary rational transduction of the "underlying" string to be parsed.

§5. carefully connects the bilexical grammar formalism of this chapter to other bilexical formalisms such as dependency, context-free, head-automaton, and link grammars. In particular, we apply the fast parsing idea to these formalisms.

The conclusions in §6. summarize the result and place it in the context of other work by the author, including a recent asymptotic improvement.

2. A SIMPLE BILEXICAL FORMALISM

The bilexical formalism developed in this chapter is modeled on dependency grammar (Gaifman, 1965; Mel'čuk, 1988). It is equivalent to the class of **split bilexical grammars** (including split bilexical CFGs and split HAGs) defined

in (Eisner and Satta, 1999). More powerful bilexical formalisms also exist, and improved parsing algorithms for these are cited in §5.6 and §5.8.

Form of the Grammar. We begin with a simple version of the formalism, to be modified later in the chapter. A [split] unweighted bilexical grammar consists of the following elements:

- A set V of words, called the (terminal) **vocabulary**, which contains a distinguished symbol ROOT.

- For each word $w \in V$, a pair of deterministic finite-state automata ℓ_w and r_w. Each automaton accepts some regular subset of V^*.

t is defined to be an upper bound on the number of states in any single automaton. (g will be defined in §4.2 as an upper bound on lexical ambiguity.)

The **dependents** of word w are the headwords of its arguments and adjuncts. Speaking intuitively, automaton ℓ_w specifies the possible sequences of left dependents for w. So these allowable sequences, which are word strings in V^*, form a regular set. Similarly r_w specifies the possible sequences of right dependents for w.

By convention, the first element in such a sequence is closest to w in the surface string. Thus, the possible dependent sequences (from left to right) are specified by $\mathcal{L}(\ell_w)^R$ and $\mathcal{L}(r_w)$ respectively. For example, if the tree shown in Figure 1.1a is grammatical, then we know that ℓ_{plan} accepts *the*, and r_{plan} accepts *of raise*.

To get fast parsing, it is reasonable to ask that the automata individually have few states (i.e., that t be small). However, we wish to avoid any penalty for having

- many (distinct) automata—two per word in V;

- many arcs leaving an automaton state—one per possible dependent in V.

That is, the vocabulary size $|V|$ should not affect performance at all.

We will use $Q(\ell_w)$ and $Q(r_w)$ to denote the state sets of ℓ_w and r_w respectively; $I(\ell_w)$ and $I(r_w)$ to denote their initial states; and predicate $F(q)$ to mean that q is a final state of its automaton. The transition functions may be notated as a single pair of functions ℓ and r, where $\ell(w, q, w')$ returns the state reached by ℓ_w when it leaves state q on an arc labeled w', and similarly $r(w, q, w')$.

Notice that as an implementation matter, if the automata are defined in any systematic way, it is not necessary to actually store them in order to represent the grammar. One only needs to choose an appropriate representation for states q and define the I, F, ℓ, and r functions.

Meaning of the Grammar. We now formally define the language generated by such a grammar, and the structures that the grammar assigns to sentences of this language.

Let a **dependency tree** be a rooted tree whose nodes (both internal and external) are labeled with words from V, as illustrated in Figure 1.1a; the root is labeled with the special symbol $\text{ROOT} \in V$. The children ('dependents') of a node are ordered with respect to each other and the node itself, so that the node has both **left children** that precede it and **right children** that follow it.

A dependency tree is grammatical iff for every word token w that appears in the tree, ℓ_w accepts the (possibly empty) sequence of w's left children (from right to left), and r_w accepts the sequence of w's right children (from left to right).

A string $\omega \in V^*$ is generated by the grammar, with analysis T, if T is a grammatical dependency tree and listing the node labels of T in infix order yields the string ω followed by ROOT. ω is called the **yield** of T.

Bilexicalism. The term *bilexical* refers to the fact that (i) each $w \in V$ may specify a wholly different choice of automata ℓ_w and r_w, and furthermore (ii) these automata ℓ_w and r_w may make distinctions among individual words that are appropriate to serve as *children* (dependents) of w. Thus the grammar is sensitive to specific *pairs* of lexical items.

For example, it is possible for one lexical verb to select for a completely idiosyncratic set of nouns as subject, and another lexical verb to select for an entirely different set of nouns. Since it never requires more than a two-state automaton (though with many arcs!) to specify the set of possible subjects for a verb, there is no penalty for such behavior in the parsing algorithm to be described here.

3. $O(n^5)$ AND $O(n^3)$ RECOGNITION

This section develops a basic $O(n^3)$ recognition method for simple bilexical grammars as defined above. We begin with a naive $O(n^5)$ method drawn from context-free 'dotted-rule' methods such as (Earley, 1970; Graham et al., 1980). Second, we will see why this method is inefficient. Finally, a more efficient $O(n^3)$ algorithm is presented.

Both methods are essentially chart parsers, in that they use dynamic programming to build up an analysis of the whole sentence from analyses of its substrings. However, the slow method combines traditional *constituents*, whose lexical heads may be in the middle, while the fast method combines what we will call *spans*, whose heads are guaranteed to be at the edge.

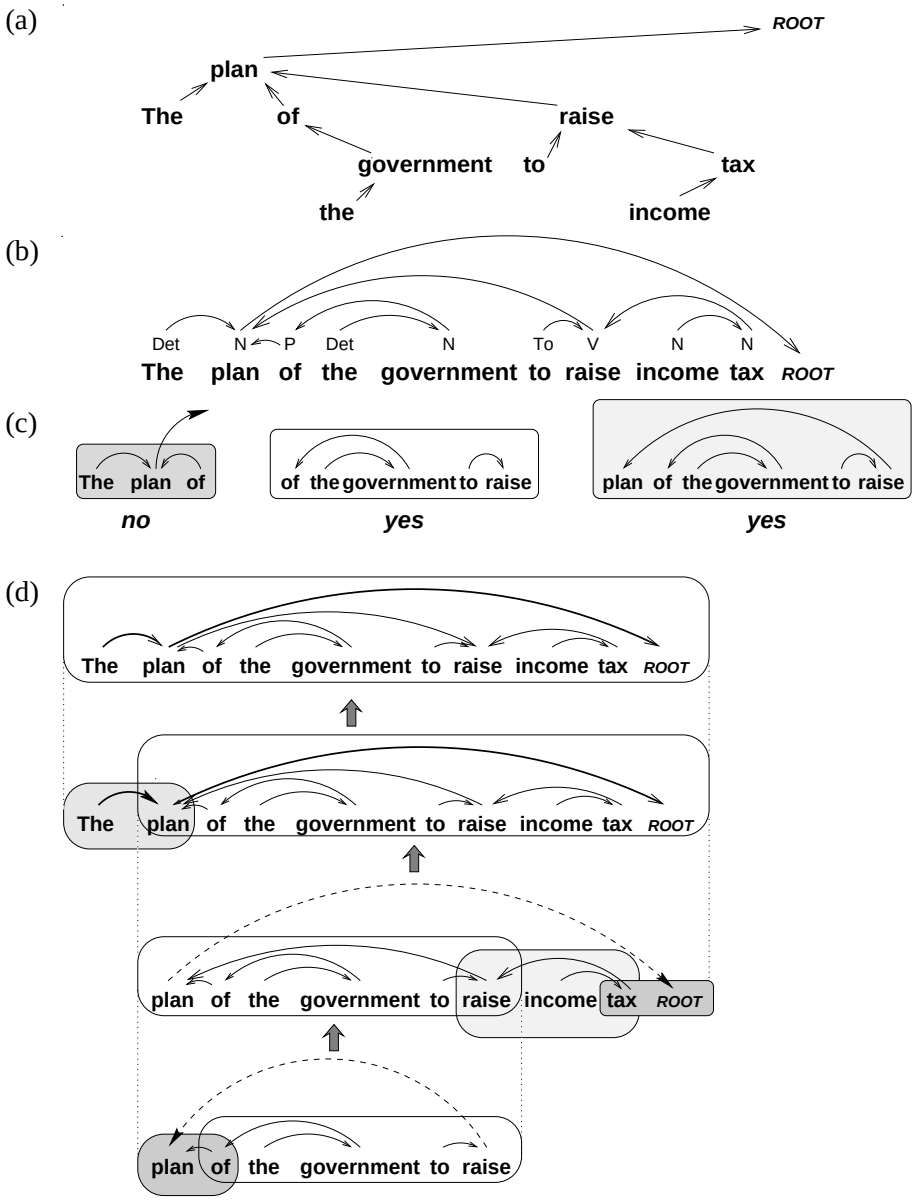


Figure 1.1 [Shading in this figure has no meaning.] (a) A dependency parse tree. (b) The same tree shown flattened out. (c) A span of the tree is any substring such that no interior word of the span links to any word outside the span. One non-span and two spans are shown. (d) A span may be decomposed into smaller spans as repeatedly shown; therefore, a span can be built from smaller spans by following the arrows upward. The parsing algorithm (Fig. 1.3–1.4) builds successively larger spans in a dynamic programming table (chart). The minimal spans, used to seed the chart, are linked or unlinked word bigrams, such as *The*→*plan* or *tax* ROOT, as shown.

3.1 NOTATION AND PRELIMINARIES

The input to the recognizer is a string of words, $\omega = w_1 w_2 \dots w_n \in V^*$. We put $w_{n+1} = \text{ROOT}$, a special symbol that does not appear in ω. For $i \leq j$, we write $w_{i,j}$ to denote the input substring $w_i w_{i+2} \dots w_j$.

Generic Chart Parsing. There may be many ways to analyze $w_{i,j}$. Each grammatical **analysis** has as its **signature** an **item**, or tuple, that concisely and completely describes its ability to combine with analyses of neighboring input substrings. Many analyses may have the same item as signature. This chapter will add some syntactic sugar and draw items as schematic pictures of analyses.

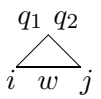
C (the **chart**) is an $(n+1) \times (n+1)$ array. The chart **cell** $C_{i,j}$ accumulates the set of signatures of all analyses of $w_{i,j}$. It must be possible to enumerate the set—or more generally, certain subsets defined by particular fixed properties—in time $O(1)$ per element.[2] In addition, it must be possible to perform an $O(1)$ duplicate check when adding a new item to a cell. A standard implementation is to maintain linked lists for enumerating the relevant subsets, together with a hash table (or array) for the duplicate check.

Analysis. If S bounds the number of items per chart cell, then the space required by a recognizer is clearly $O(n^2 S)$. The time required by the algorithms we consider is $O(n^3 S^2)$, because for each of the $O(n^3)$ values of i, j, k such that $1 \leq i \leq j < k \leq n+1$, they will test each of the $\leq S$ items in $C_{i,j}$ against each of the $\leq S$ items in $C_{j+1,k}$, to see whether analyses with those items as signatures could be grammatically combined into an analysis of $w_{i,k}$.

Efficiency therefore requires keeping S small. The key difference between the $O(n^5)$ method and the $O(n^3)$ method will be that S is $O(n)$ versus $O(1)$.

3.2 NAIVE BILEXICAL RECOGNITION

An Algorithm. The obvious approach for bilexical grammars is for each analysis to represent a subtree, just as for an ordinary CFG. More precisely, each analysis of $w_{i,j}$ is a kind of **dotted subtree** that may not yet have acquired all its children.[3] The signature of such a dotted subtree is an item (w, q_1, q_2). This may be depicted more visually as



where $w \in w_{i,j}$ is the **head word** at the root of the subtree, $q_1 \in Q(\ell_w)$, and $q_2 \in Q(r_w)$. If both q_1 and q_2 are final states, then the analysis is a complete constituent.

The resulting algorithm is specified declaratively using sequents in Figure 1.2a–b, which shows how the items combine.

Analysis. It is easy to see from Figure 1.2a that each chart cell $C_{i,j}$ can contain $S = O(\min(n, |V|)t^2)$ possible items: there are $O(\min(n, |V|))$ choices for w, and $O(t)$ choices for each of q_1 and q_2 once w is known. It follows that the runtime is $O(n^3 S^2) = O(n^3 \min(n, |V|)^2 t^4)$.

More simply and generally, one can find the runtime by examining Figure 1.2b and seeing that there are $O(n^3 \min(n, |V|)^2 t^4)$ ways to instantiate the four rule templates. Each is instantiated at most once and in $O(1)$ time. (McAllester, 1999) proves that with appropriate indexing of items, this kind of runtime analysis is correct for a very general class of algorithms specified declaratively by inference rules.

An Improvement. It is possible to reduce the t^4 factor to just t, since each attachment decision really depends only on one state (at the parent), not four states. This improved method is shown in Figure 1.2c. It groups complete constituents together under a single item even if they finished in different final states—a trick we will be using again.

Note that the revised method always attaches right children before left children, implying that a given dependency tree is only derived in one way. This property is important if one wishes to enhance the algorithm to compute the total number of distinct trees for a sentence, or their total probability, or related quantities needed for the Inside-Outside estimation algorithm.

Discussion. Even with the improvement, parsing is still $O(n^5)$ (for $n < |V|$). Why so inefficient? Because there are too many distinct possible signatures. Whether LINK-L can make one tree a new child of another tree depends on the head words of both trees. Hence signatures must mention head words. Since the head word of a tree that analyzes $w_{i,j}$ could be any of the words $w_i, w_{i+1}, \ldots w_j$, and there may be n distinct such words in the worst case (assuming $n < |V|$), the number S of possible signatures for a tree is at least n.

In more concrete terms, the problem is that each chart cell may have to maintain many differently-headed analyses of the same string. Chomsky's noun phrase *visiting relatives* has two analyses: a kind of relatives vs. a kind of visiting. A bilexical grammar knows that only the first is appropriate in the context *hug visiting relatives*, and only the second is appropriate in the context *advocate visiting relatives*. So the two analyses must be kept separate in the chart: they will combine with context differently and therefore have different signatures.

3.3 EFFICIENT BILEXICAL RECOGNITION

Constituents vs. Spans. To eliminate these two additional factors of n, we must reduce the number of possible signatures for an analysis. The solution is for analyses to represent some kind of contiguous string other than constituents.

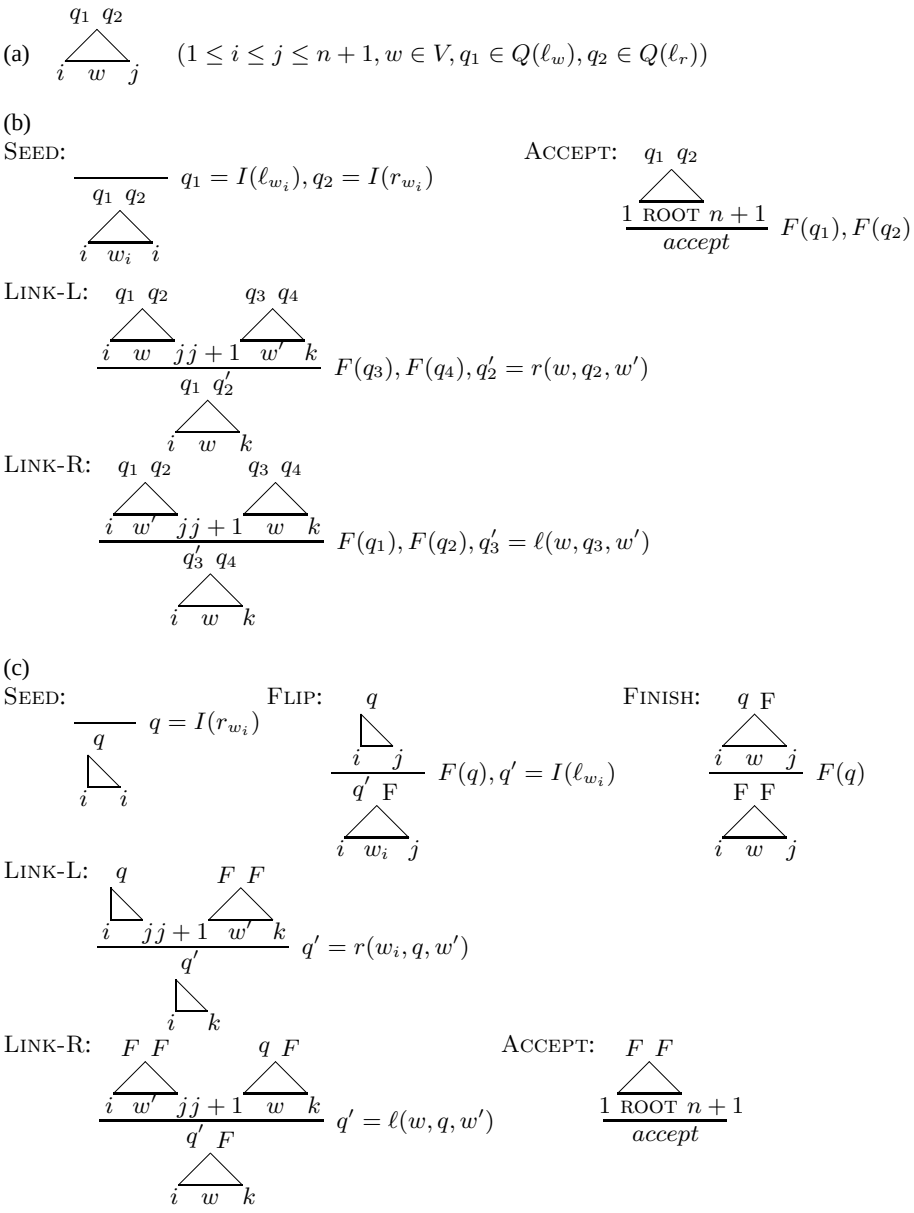


Figure 1.2 Declarative specification of an $O(n^5)$ algorithm. (a) Form of items in the parse chart. (b) Inference rules. The algorithm can derive an analysis with the signature below ——— by combining analyses with the signatures above ———, provided that the input and grammar satisfy any properties listed to the right of ———. (c) A variant that reduces the grammar factor from t^4 to t. F is a literal that means 'an unspecified final state.'

Each analysis in $C_{i,j}$ will be a new kind of object called a **span**, which consists of one or two 'half-constituents' in a sense to be described. The headword(s) of a span in $C_{i,j}$ are *guaranteed* to be at positions i and/or j in the sentence. This guarantee means that where $C_{i,j}$ in the previous section had up to n-fold uncertainty about the location of the headword of $w_{i,j}$, here it will have only 3-fold uncertainty. The three possibilities are that w_i is a headword, that w_j is, or that both are.

Given a dependency tree, we know what its constituents are: a constituent is any substring consisting of a word and all its descendants. The inefficient parsing algorithm of §3.2 assembled the correct tree by finding and gluing together analyses of the tree's (dotted) constituents in an approved way. For something similar to be possible with spans, we must define what the spans of a given dependency tree are, and how to glue analyses of spans together into analyses of larger spans. Not every substring of the sentence is a constituent of this (or any) sentence's correct parse, and in the same way, not every substring is a span of this (or any) sentence's correct parse.

Definition of Spans. Figure 1.1a–c illustrates what spans are. A span of the dependency tree in (a) and (b) is any substring $w_{i,j}$ of the input such that none of the interior words of the span communicate with any words outside the span. Formally: if $i < k < j$, and w_k is a child or parent of $w_{k'}$, then $i \leq k' \leq j$.

Thus, just as a constituent links to the rest of the sentence only through its head word, which may be located anywhere in the constituent, a span $w_{i,j}$ links to the rest of the sentence only through its **endwords** w_i and w_j, which are located at the edges of the span. We call $w_{i+1,j-1}$ the span's **interior**.

Assembling Spans. Since we will build the parse by assembling possible spans, and the interiors of adjacent spans are insulated from each other, we crucially are allowed to forget the internal analysis of a span once we have built it. When we combine two adjacent such spans, we never add a link from or to the interior of either. For, by the definition of span, if such a link were necessary, then the spans being combined could not be spans of the true parse anyway. There is always some other way of decomposing the true parse (itself a span) into smaller spans so that no such links from or to interiors are necessary.

Figure 1.1d shows such a decomposition. Any span analysis of more than two words, say $w_{i,k}$, can be decomposed uniquely by the following deterministic procedure. Choose j such that w_j is the rightmost word in the interior of the span ($i < j < k$) that links to or from w_i; if there is no such word, put $j = i + 1$. Because crossing links are not allowed in a dependency tree—a property known as **projectivity**—the substrings $w_{i,j}$ and $w_{j,k}$ must also be spans. We can therefore assemble the original $w_{i,k}$ analysis by concatenating the $w_{i,j}$ and $w_{j,k}$ spans, and optionally adding a link between the endwords,

w_i and w_k. By construction, there is never any need to add a link between any other pair of words. Notice that when the two narrower spans are concatenated, w_j gets its left children from one span and its right children from the other, and will never be able to acquire additional children since it is now span-internal.

By our choice of j, the left span in the concatenation, $w_{i,j}$, is always **simple** in the following sense: it has a direct link between w_i and w_j, or else has only two words. ($w_{i,k}$ is decomposed at the maximal j such that $i < j < k$ and $w_{i,j}$ is simple.) Requiring the left span to be simple assures a unique decomposition (see §3.2 for motivation); the right span need not be simple.

Signatures of Spans. A span's signature needs to record only a few pertinent facts about its internal analysis. It has the form shown in Figure 1.3a. i, j indicate that the span is an analysis of $w_{i,j}$. q_1 is the state of r_{w_i} after it has read the sequence of w_i's right children that appear in $w_{i+1,j}$, and q_2 is the state of ℓ_{w_j} after it has read the sequence of w_j's left children that appear in $w_{i,j-1}$. b_1 and b_2 are bits that indicate whether w_i and w_j, respectively, have parents within $w_{i,j}$. Finally, s is a bit indicating whether the span is simple in the sense described above.

The signature must record q_1 and q_2 so that the parser knows what additional dependents w_i or w_j can acquire. It must record b_1 and b_2 so that it can detect whether such a link would jeopardize the tree form of the dependency parse (by creating multiple parents, cycles, or a disconnected graph). Finally, it must record s to ensure that each distinct analysis is derived in at most one way.

It is useful to note the following four possible types of span:

- $b_1 = b_2 = 0$. Example: *of the government to raise* in Figure 1.1c. In this case, the endwords w_i and w_j are not yet connected to each other: that is, the path between them in the final parse tree will involve words outside the span. The span consists of two 'half-constituents'—w_i with all its right descendants, followed by w_j with all its left descendants.

- $b_1 = 0, b_2 = 1$. Example: *plan of the government to raise* in Figure 1.1c. In this case, w_j is a descendant of w_i via a chain of one or more leftward links within the span itself. The span consists of w_i and all its right descendants within $w_{i+1,j}$. (w_i or w_j or both may later acquire additional right children to the right of w_j.)

- $b_1 = 1, b_2 = 0$. Example: the whole sentence in Figure 1.1b. This is the mirror image of the previous case.

- $b_1 = 1, b_2 = 1$. This case is impossible, for then some word interior to the span would need a parent outside it. We will never derive any analyses with this signature.

(a) $\begin{matrix} b_1 & & b_2 \\ q_1 & \boxed{\ s\ } & q_2 \\ i & & j \end{matrix}$ $(1 \le i < j \le n+1, q_1 \in Q(r_{w_i}) \cup \{\mathrm{F}\}, q_2 \in Q(\ell_{w_j}) \cup \{\mathrm{F}\}, b_1, b_2, s \in \{0,1\}, \neg(q_1 = \mathrm{F} \wedge q_2 = \mathrm{F}), \neg(b_1 \wedge b_2))$

(b) SEED:

$$\frac{}{\begin{matrix} 0 & & 0 \\ q_1 & \boxed{\ 1\ } & q_2 \\ i & & i+1 \end{matrix}}\ q_1 = I(r_{w_i}), q_2 = I(\ell_{w_{i+1}})$$

COMBINE:

$$\frac{\begin{matrix} b_1 & & b_2 & \neg b_2 & & b_3 \\ q_1 & \boxed{\ 1\ } & \mathrm{F} & \mathrm{F} & \boxed{\ s\ } & q_3 \\ i & & j & j & & k \end{matrix}}{\begin{matrix} b_1 & & b_3 \\ q_1 & \boxed{\ 0\ } & q_3 \\ i & & k \end{matrix}}$$

OPT-LINK-L:

$$\frac{\begin{matrix} 0 & & 0 \\ q_1 & \boxed{\ s\ } & q_2 \\ i & & j \end{matrix}}{\begin{matrix} 0 & & 1 \\ q_1' & \boxed{\ 1\ } & q_2 \\ i & & j \end{matrix}}\ \begin{matrix} q_1 \neq \mathrm{F}, q_2 \neq F, \\ q_1' = r(w_i, q_1, w_j) \end{matrix}$$

OPT-LINK-R:

$$\frac{\begin{matrix} 0 & & 0 \\ q_1 & \boxed{\ s\ } & q_2 \\ i & & j \end{matrix}}{\begin{matrix} 1 & & 0 \\ q_1 & \boxed{\ 1\ } & q_2' \\ i & & j \end{matrix}}\ \begin{matrix} q_1 \neq \mathrm{F}, q_2 \neq F, \\ q_2' = \ell(w_j, q_2, w_i) \end{matrix}$$

SEAL-L:

$$\frac{\begin{matrix} b_1 & & b_2 \\ q_1 & \boxed{\ s\ } & q_2 \\ i & & j \end{matrix}}{\begin{matrix} b_1 & & b_2 \\ \mathrm{F} & \boxed{\ s\ } & q_2 \\ i & & j \end{matrix}}\ q_1 \neq \mathrm{F}, q_2 \neq \mathrm{F}, F(q_1)$$

SEAL-R:

$$\frac{\begin{matrix} b_1 & & b_2 \\ q_1 & \boxed{\ s\ } & q_2 \\ i & & j \end{matrix}}{\begin{matrix} b_1 & & b_2 \\ q_1 & \boxed{\ s\ } & \mathrm{F} \\ i & & j \end{matrix}}\ q_1 \neq \mathrm{F}, q_2 \neq \mathrm{F}, F(q_2)$$

ACCEPT:

$$\frac{\begin{matrix} 1 & & 0 \\ \mathrm{F} & \boxed{\ s\ } & q_2 \\ 1 & & n+1 \end{matrix}}{accept}\ F(q_2)$$

Figure 1.3 Declarative specification of an $O(n^3)$ algorithm. (a) Form of items in the parse chart. (b) Inference rules. As in Fig. 1.2b, F is a literal that means 'an unspecified final state.'

```
1.  for i := 1 to n
2.      s := the item for w_{i,i+1} produced by SEED
3.      Discover(i, i+1, s)
4.      Discover(i, i+1, OPT-LINK-L(s))
5.      Discover(i, i+1, OPT-LINK-R(s))
6.  for width := 2 to n
7.      for i := 1 to (n+1) − width
8.          k := i + width
9.          for j := i+1 to k − 1
10.             foreach simple item s1 in C^L_{i,j}
11.                 foreach item s2 in C^R_{j,k} such that COMBINE(s1, s2) is defined
12.                     s := COMBINE(s1, s2)
13.                     Discover(i, k, s)
14.                     if OPT-LINK-L(s) and OPT-LINK-R(s) are defined
15.                         Discover(i, k, OPT-LINK-L(s))
16.                         Discover(i, k, OPT-LINK-R(s))
17. foreach item s in C^R_{1,n+1}
18.     if ACCEPT(s) is defined
19.         return accept
20. return reject
```

Figure 1.4 Pseudocode for an $O(n^3)$ recognizer. The functions in small caps refer to the (deterministic) inference rules of Figure 1.3. Discover(i, j, s) adds SEAL-L(s) (if defined) to $C^R_{i,j}$ and SEAL-R(s) (if defined) to $C^L_{i,j}$.

The Span-Based Algorithm. A declarative specification of the algorithm is given in Figure 1.3, which shows how the items combine. The reader may choose to ignore s for simplicity, since the unique-derivation property may speed up recognition but does not affect its correctness. For concreteness, pseudocode is given in Figure 1.4.

The SEED rule seeds the chart with the minimal spans, which are two words wide. COMBINE is willing to combine two spans if they overlap in a word w_j that gets all its left children from the left span (hence 'F' appears in the rule), all its right children from the right span (again 'F'), and its parent in exactly one of the spans (hence '$b_2, \neg b_2$'). Whenever a new span is created by seeding or combining, the OPT-LINK rules can add an optional link between its endwords, provided that neither endword already has a parent.

The SEAL rules check that an endword's automaton has reached a final (accepting) state. This is a precondition for COMBINE to trap the endword in the interior of a larger span, since the endword will then be unable to link to any more children. While COMBINE could check this itself, using SEAL is asymptotically more efficient because it conflates different final states into a single item—exactly as FINISH did in Figure 1.2c.

Analysis. The time requirements are $O(n^3t^2)$, since that is the number of ways to instantiate the free variables in the rules of Figure 1.3b (McAllester, 1999). As t is typically small, this compares favorably with $O(n^5t)$ for the naive algorithm of §3.2. Even better, §3.4 will obtain a speedup to $O(n^3t)$.

The space requirements are naively $O(n^2t^2)$, since that is the number of ways to instantiate the free variables in Figure 1.3a, i.e., the maximum number of items in the chart. The pseudocode in Figure 1.4 shows that this can be reduced to $O(n^2t)$ by storing only items for which $q_1 = \mathrm{F}$ or $q_2 = \mathrm{F}$ (in separate charts C^R and C^L respectively). The other items need not be added to any chart, but can be fed to the OPT-LINK and SEAL rules immediately upon creation, and then destroyed.

3.4 AN ADDITIONAL $O(t)$ SPEEDUP

The above algorithm can optionally be sped up from $O(n^3t^2)$ to $O(n^3t)$, at the cost of making it perhaps slightly harder to understand.

Every item in Figure 1.3 has either 0 or 1 of the states q_1, q_2 instantiated as the special symbol F. We will now modify the algorithm so that either 1 or 2 of those states are always instantiated as F (except in items produced by SEED). This is possible because q_2 does not really matter in OPT-LINK-L, nor does q_1 in OPT-LINK-R. The payoff is that these rules, as well as COMBINE, will only need to consider one state at a time.

All that is necessary is to modify the applicability conditions of the inference rules. COMBINE gets the additional condition $q_1 = \mathrm{F} \vee q_3 = \mathrm{F}$. OPT-LINK-L and SEAL-L drop the condition that $q_2 \neq \mathrm{F}$, while OPT-LINK-R and SEAL-R drop the condition that $q_1 \neq \mathrm{F}$.

To preserve the property that derivations are unique, two additional modifications are now necessary. To eliminate the freedom to apply SEAL either before or after COMBINE, the SEAL rules should be restricted to apply only to simple spans (i.e., $s = 1$). And to eliminate the freedom to apply both SEAL-L and SEAL-R in either order to the output of SEED, the SEAL-L rule should require that $q_2 \neq \mathrm{F} \vee b_2 = 1$.

4. VARIATIONS

In this section, we describe useful modifications that may be made to the formalism and/or the algorithm above.

4.1 WEIGHTED GRAMMARS

The ability of a verb to subcategorize for an idiosyncratic set of nouns, as above, can be used to implement black-and-white ('hard') selectional restrictions. Where bilexical grammars are really useful, however, is in capturing

gradient ('soft') selectional restrictions. A **weighted bilexical grammar** can equip each verb with an idiosyncratic *probability distribution* over possible object nouns, or indeed possible dependents of any sort. We now formalize this notion.

Weighted Automata. A **weighted DFA**, A, is a deterministic finite-state automaton that associates a real-valued **weight** with each arc and each final state (Mohri et al., 1996). Following heavily-weighted arcs is intuitively 'good,' 'probable,' or 'common'; so is stopping in a heavily-weighted final state. Each accepting path through A is automatically assigned a weight, namely, the sum of all arc weights on the path and the final-state weight of the last state on the path. Each string α accepted by A is assigned the weight of its accepting path.

Weighted Grammars. Now, we may define a weighted bilexical grammar as a bilexical grammar in which all the automata ℓ_w and r_w are weighted DFAs. We define the weight of a dependency tree under the grammar as the sum, over all word tokens w in the tree, of the weight with which ℓ_w accepts w's sequence of left children plus the weight with which r_w accepts w's sequence of right children.

Given an input string ω, the **weighted parsing problem** is to find the highest-weighted grammatical dependency tree whose yield is ω.

From Recognition to Weighted Parsing. One may turn the recognizer of §3.3 into a parser in the usual way. Together with each item stored in a chart cell $C_{i,j}$, one must also maintain the highest-weighted known analysis with that item as signature, or a parse forest of all known analyses with that signature. In the implementation, items may be mapped to analyses via a hash table or array.

When we apply a rule from Figure 1.3b to derive a new item from old ones, we must also derive an associated analysis (or forest of analyses), and the weight of this analysis if the grammar is weighted.

When parsing, how should we *represent* an analysis of a span? (For comparison, an analysis of a constituent can be represented as a tree.) A general method is simply to store the span's derivation: we may represent any analysis as a copy of the rule that produced it together with pointers to the analyses that serve as inputs (i.e., antecedents) to that rule. Or similarly, one may follow the decomposition of §3.3 and Figure 1.1d. Then an analysis of $w_{i,k}$ is a triple $(\alpha, \beta, linktype)$, where α points to an analysis of a simple span $w_{i,j}$, β points to an analysis of a span $w_{j,k}$, and $linktype \in \{\leftarrow, \rightarrow, \text{NONE}\}$ specifies the direction of the link (if any) between w_i and w_k. In the base case where $k = i+1$, then α and β instead store w_i and w_k respectively.

We must also know how to compute the weight of an analysis. Any convenient definition will do, so long as the weight of a full parse comes out correctly.

In all cases, we will define the weight of an analysis produced by a rule to be the total weight of the input(s) to that rule, plus another term derived from the conditions on the rule. For SEED and COMBINE, the additional term is 0; for OPT-LINK-L or OPT-LINK-R, it is the weight of the transition to q_1' or q_2' respectively; for SEAL-L, SEAL-R, or ACCEPT, it is the final-state weight of q_1, q_2, or q_2 respectively.

As usual, the strategy of maintaining only the highest-weighted analysis of each signature works because context-free parsing has the **optimal substructure property**. That is, any *optimal* analysis of a long string can be found by gluing together just *optimal* analyses of shorter substrings. For suppose that a and a' are analyses of the same substring, and have the same signature, but a has less weight than a'. Then suboptimal a cannot be part of any optimal analysis b in the chart—for if it were, the definition of signature ensures that we could substitute a' for a in b to get an analysis b' of greater total weight than b and the same signature as b, which contradicts b's optimality.

4.2 POLYSEMY

We now extend the formalism to deal with lexical selection. Regrettably, the input to a parser is typically not a string in V^*. Rather, it contains ambiguous tokens such as *bank*, whereas the 'words' in V are word senses such as *bank*$_1$, *bank*$_2$, and *bank*$_3$, or part-of-speech-tagged words such as *bank*/N and *bank*/V. If the input is produced by speech recognition or OCR, even more senses are possible.

One would like a parser to resolve these ambiguities simultaneously with the structural ambiguities. This is particularly true of a bilexical parser, where a word's dependents and parent provide clues to its sense and vice-versa.

Confusion Sets. We may modify the formalism as follows. Consider the unweighted case first. Let Ω be the real input—a string not in V^* but rather in $\mathcal{P}(V)^*$, where $\mathcal{P}$ denotes powerset. Thus the ith symbol of Ω is a **confusion set** of possibilities for the ith word of the input, e.g., $\{$*bank*$_1$, *bank*$_2$, *bank*$_3\}$. Ω is generated by the grammar, with analysis T, if some string $\omega \in V^*$ is so generated, where ω is formed by replacing each set in Ω with one of its elements. Note that the yield of T is ω, not Ω.

For the weighted case, each confusion set in the input string Ω assigns a weight to each of its members. Again, intuitively, the heavily-weighted members are the ones that are commonly correct, so the noun *bank*/N would be weighted more highly than the verb *bank*/V. We score parses as before, except that now we also add to a dependency tree's score the weights of all the words that label its nodes, as selected from their respective confusion sets. Formally, we say that $\Omega = W_1 W_2 \ldots W_n \in \mathcal{P}(V)^*$ is generated by the grammar, with analysis T and weight $\mu_T \mu_1 + \cdots + \mu_n$, if some string $\omega = w_1 w_2 \ldots w_n \in V^*$

is generated with analysis T of weight μ_T, and for each $1 \leq i \leq n$, ω_i appears in the set W_i with weight μ_i.

Modifying the Algorithm. Throughout the algorithm of Figure 1.3, we must replace each integer i (similarly j, k) with a pair of the form (i, w_i), where $w_i \in W_i$. That ensures that the signature of an analysis of $W_{i,j}$ will record the senses w_i and w_j of its endwords. The OPT-LINK rules refer to these senses when determining whether w_j can be a child of w_i or vice-versa. Moreover, COMBINE now requires its two input spans to agree not only on j but also on the sense w_j of their overlapping word W_j, so that this word's left children, right children, and parent are all appropriate to the same sense. The SEED rule nondeterministically chooses senses $w_i \in W_i$ and $w_{i+1} \in W_{i+1}$; to avoid double-counting, the weight of the resulting analysis is taken to be the weight with which w_i appears in W_i only.

If g is an upper bound on the size of a confusion set, then these modifications multiply the algorithm's space requirements by $O(g^2)$ and its time requirements by $O(g^3)$.

4.3 STRING-LOCAL CONSTRAINTS

When the parser is resolving polysemy as in §4.2, it can be useful to implement string-local constraints. The SEED rule may be modified to disallow an arbitrary list of word-sense bigrams $w_i w_{i+1}$. More usefully, it may be made to favor some bigrams over others by giving them higher weights. Then the sense of one word will affect the preferred sense of adjacent words. (This is in addition to affecting the preferred sense of the words it links to).

For example, suppose each word is polysemous over several part-of-speech tags, which the parser must disambiguate. A useful hack is to define the weight of a parse as the log-probability of the parse, as usual, *plus* the log-probability of its tagged yield under the trigram tagging model of (Church, 1988). Then a highly-weighted parse will tend to be one whose tagged dependency structure and string-local structure are simultaneously plausible. This has been shown useful for probabilistic systems that simultaneously optimize tagging and parsing (Eisner, 1996a). (See (Lafferty et al., 1992) for a different approach.)

To add in the trigram log-probability in this way, regard each input word as a confusion set W_i whose elements have the form $w_i = (v_i, t_i, t_{i+1})$. Here each v_i is an ordinary word (or sense) and t_i, t_{i+1} are hypothesized part-of-speech tags for v_i, v_{i+1} respectively. Now SEED should be restricted to produce only word-sense bigrams $(v_i, t_i, t_{i+1})(v_{i+1}, t_{i+1}, t_{i+2})$ that agree on t_{i+1}. The score of such a bigram is $\log \Pr(v_i \mid t_i) + \log \Pr(t_i \mid t_{i+1}, t_{i+2})$. (If $i = 1$, it is also necessary to add $\log \Pr(\text{STOP} \mid t_1, t_2)$.) Notice that (for notational convenience) we are treating the word sequence as generated from right to left, not vice-versa.

4.4 RATIONAL TRANSDUCTIONS

Polysemy (§4.2) and string-local constraints (§4.3) are both simple, local string phenomena that are inconvenient to model within the bilexical grammar. Many other such phenomena exist in language: they tend to be morphological in nature and easily modeled by finite-state techniques that apply to the yield of the dependency tree. This section conveniently extends the formalism and algorithm to accommodate such techniques. The previous two sections are special cases.

Underlying and Surface Strings. We distinguish the "underlying" string $\omega = w_1 w_2 \ldots w_n \in V^*$ from the "surface" string $\Omega = W_1 W_2 \ldots W_N \in X^*$. Thus V is a collection of morphemes (word senses), whereas X is typically a collection of graphemes (orthographic words). It is not necessary that $n = N$.

It is the underlying string ω that is described by the bilexical grammar. In general, ω is related to our input Ω by a possibly nondeterministic, possibly weighted finite-state transduction R (Mohri et al., 1996), as defined below.

We say that the surface string Ω is grammatical, with analysis (T, P), if T is a dependency parse tree whose fringe, ω ROOT, is transduced to Ω along an accepting path P in R. Notice that the analysis describes the tree, the underlying string, and the alignment between the underlying and surface strings.

The weighted parsing problem is now to reconstruct the best analysis (T, P) of Ω. The weight of an analysis is the weight of T plus the weight of P. For example, if weights are defined to be log-probabilities under a generative model, then the weight of T is the log-probability of stochastically generating the parse tree T and then stochastically transducing its fringe to the observed input.

Linguistic Uses. The transducer R may be used for many purposes. It can map different senses onto the same grapheme (polysemy) or vice-versa (spelling variation, contextual allomorphy). If the output alphabet X consists of letters rather than words, the transducer can apply morphological rules, such as the affixation and spelling rule in *try -ed* $\rightarrow$ *tried* (Koskenniemi, 1983; Kaplan and Kay, 1994). It can also perform more interesting kinds of local morphosyntactic processes (*PAST TRY* $\rightarrow$ *try -ed* (affix hopping), *NOT CAN* $\rightarrow$ {*can't, cannot*}, *PRO* $\rightarrow \epsilon$, ". $\rightarrow$.").

In another vein, R may be an interestingly weighted version of the identity transducer. This can be used to favor or disfavor local patterns in the underlying string ω. A classic example is the "that-trace" filter. Similarly, the trigram model of §4.3 can be implemented easily with a transducer that merely removes the tags from tagged words, and whose weights are given by log-probabilities under a trigram model.

Finally, if R is used to describe a stochastic noisy channel that has corrupted or translated the input in some way, then the parser will automatically correct

for the noise. Most ambitiously, R could be a generative acoustic model, and X an an alphabet of acoustic observations. In this case, the bilexical grammar would essentially be serving as the language model for a speech recognizer.

It is often convenient to define R as a composition of several simpler weighted transducers (Mohri et al., 1996), each of which handles just one of the above phenomena. For example, in order to map a sequence of abstract morphemes and punctuation tokens ($\in V*$) to a sequence of ASCII characters ($\in X*$), one could use the following transducer cascade: affix hopping, "that-trace" penalization, followed by deletion of phonological nulls, then conventional processes such as capitalization marking and comma absorption, then realization of abstract morphemes as lemmas or null strings, then various morphological rules, and finally a stochastic model of typographical errors. Given some text Ω that is supposed to have emerged from this pipeline, the parser's job is to find a plausible way of renormalizing it that leads to a good parse.

Transducer Notation. The **finite-state transducer** R has the same form as a (nondeterministic) finite-state automaton. However, the arcs are labeled not by symbols $w \in V$ but rather by pairs $\gamma : \Gamma$, where $\gamma \in V^*$ and $\Gamma \in X^*$.

The transducer R is said to **transduce** γ to Γ along path P if the arcs of P are consecutively labeled $\gamma_1 : \Gamma_1, \gamma_2 : \Gamma_2, \ldots \gamma_k : \Gamma_k$, and $\gamma_1\gamma_2 \cdots \gamma_k = \gamma$ and $\Gamma_1\Gamma_2 \cdots \Gamma_k = \Gamma$. We call this transduction **terminal** if $\gamma_k = \gamma$ (or $k = 0$).

One says simply that R transduces ω to Ω if it does so along an **accepting path**, i.e., a path from the initial state of R to a final state. The path's weight can be defined as in §4.1, in terms of weights on the arcs and final states of R.

We may assume without loss of generality that the strings γ have length ≤ 1. That is, all arc labels have the form $\overline{w} : \Gamma$ where $\overline{w} \in V \cup \{\epsilon\}$ and $\Gamma \in X^*$.

We reuse the notation of §2. as follows. $Q(R)$ and $I(R)$ denote the set of states and the initial state of R, and the predicate $F(r)$ means that state $r \in Q(R)$ is final. The transition predicate $R(r, r', \overline{w} : \Gamma)$ is true if there is an arc from r to r' with label $\overline{w} : \Gamma$. Its ϵ-left-closure $R^*(r, r', \overline{w} : \Gamma)$ is true iff R terminally transduces $\overline{w}$ to Γ along some path from r to r'.

Modifying the Inference Rules. Recall that when modifying the algorithm to handle polysemy, we replaced each integer i in Figure 1.3 with a pair (i, w). For the more general case of transductions, we similarly replace i with a triple (i, w, r), where $w \in V, r \in Q(R)$. An item of the form

$$\begin{array}{ccc} \cdots & & \cdots \\ \cdots & \boxed{\cdots} & \cdots \\ i, w, r & & j, w', r' \end{array} \quad (0 \leq i \leq j \leq n;\ w, w' \in V;\ r, r' \in Q(R); \cdots)$$

represents the following hypothesis about the correct sentential analysis (T, P): that the tree T has a span $w\beta w'$ (for some string β) such that $\beta w'$ is terminally transduced to the surface substring $W_{i+1,j}$ along a subpath of P from state r to

(a) SEED: $\dfrac{R^*(r, r', w' : W_{i+1,j})}{\overset{0}{q_1}\boxed{1}\overset{0}{q_2}\quad i, w, r \quad j, w', r'}\ q_1 = I(r_{w_i}), q_2 = I(\ell_{w_{i+1}})$

ACCEPT: $\dfrac{R^*(I(R), r, w : W_{1,i}) \quad \overset{1}{\mathrm{F}}\boxed{s}\overset{0}{q_2}\ i, w, r \quad j, \textsc{Root}, r' \quad R^*(r', r'', \epsilon : W_{j+1,n})}{accept}\ F(q_2), F(r'')$

(b) FINAL-w: $\dfrac{}{R^*(r, r', w : W_{i+1,j})}\ R(r, r', w : W_{i+1,j})$ FINAL-ϵ: $\dfrac{}{R^*(r, r, \epsilon : W_{i+1,i})}$

EXT-LEFT: $\dfrac{R^*(r', r'', \overline{w} : W_{j+1,k})}{R^*(r, r'', \overline{w} : W_{i+1,k})}\ R(r, r', \epsilon : W_{i+1,j})$

(c)

START-PREFIX: $\dfrac{R^*(I(R), r, w : W_{1,i})}{\xrightarrow{i,w} r}$ EXT-PREFIX: $\dfrac{\xrightarrow{i,w} r \quad R^*(r, r', w' : W_{i+1,j})}{\xrightarrow{j,w'} r'}$

START-SUFFIX: $\dfrac{R^*(r, r', \epsilon : W_{i+1,n})}{r \xrightarrow{i}}\ F(r')$ EXT-SUFFIX: $\dfrac{R^*(r, r', w' : W_{i+1,j}) \quad r' \xrightarrow{j}}{r \xrightarrow{i}}$

(d) SEED: $\dfrac{\xrightarrow{i,w} r \quad R^*(r, r', w' : W_{i+1,j}) \quad r' \xrightarrow{j}}{\overset{0}{q_1}\boxed{1}\overset{0}{q_2}\quad i, w, r \quad j, w', r'}\ q_1 = I(r_{w_i}), q_2 = I(\ell_{w_{i+1}})$

(e)
```
1.  Agenda := {}                                   (* priority queue of items by weight of their associated derivations *)
2.  Done := {}                                     (* set of items indexed as discussed in §3.1, §3.2 *)
3.  foreach x that can be produced by a rule with no inputs
4.      AddAgenda(x, Agenda)                       (* if duplicate, then also removes copy with the lighter derivation *)
5.  while Agenda ≠ {}
6.      x := Pop(Agenda)                           (* highest-weighted item *)
7.      if x = accept then return accept           (* also return associated derivation *)
8.      if x ∉ Done
9.          AddDone(x, Done)                       (* updates indices appropriately *)
10.         foreach rule r
11.             if r(x) is defined then AddAgenda(r(x), Agenda)        (* as above *)
12.             foreach z ∈ ⋃_{y∈Done} {(x, y), (y, x)} with r(z) defined   (* use indices *)
13.                 AddAgenda(r(z), Agenda)        (* as above *)
14. return reject
```

Figure 1.5 All non-trivial changes to Figure 1.3 needed for handling transductions of the input. (a) The minimal modification to ensure correctness. The predicate $R^*(r, r', w' : W_{i+1,j})$ is used here as syntactic sugar for an item $[r, r', w', i+1, j]$ (where $i \leq j$) that will be derived iff the predicate is true. (b) Rules for deriving those items during preprocessing of the input. (c) Deriving "forward-backward" items during preprocessing. (d) Adding "forward-backward" antecedents to parsing to rule out items that are impossible in context. (e) Generic pseudocode for agenda-based parsing from inference rules. Line 12 uses indices on y to enumerate z efficiently.

state r'.[4] Notice that if $i = j$ then $W_{i+1,j} = \epsilon$ by definition. Also notice that no claim is made about the relation of w to $W_{1,i}$ (but see below).

COMBINE must be modified along the same lines as for polysemy: it must require its two input spans to agree not only on j but on the entire triple (j, w', r'). As before, OPT-LINK should be defined in terms of the underlying words w, w'.

It is only the SEED and ACCEPT rules that actually need to examine the transducer R. Modified versions are shown in Figure 1.5a. These rules make reference to the ϵ-left-closed transition relation $R^*(\cdots)$, which Figure 1.5b shows how to precompute on substrings of the input Ω.

From Recognition to Parsing. This modified recognition algorithm yields a parsing algorithm just as in §4.1. An analysis with the signature shown above has two parts: an analysis of the span $w\beta w'$, and the r-to-r' subpath that terminally transduces $\beta w'$ to $W_{i+1,j}$. Its weight is the sum of the weights of these two parts. To compute this weight, each rule in Figure 1.5a–b should define the weight of its output to be the total weight of its inputs, plus the arc or final-state weight associated with any $R(r, r', \ldots)$ or $F(\cdots)$ that it tests.

Cyclic Derivations. If R can transduce non-empty underlying substrings to ϵ, we must now use chart cells $C_{i,i}$, for spans that correspond to the surface substring $W_{i+1,i} = \epsilon$. In the general case where R can do so along cyclic paths, so that such spans may be unbounded, items can no longer be combined in a fixed order as in Figure 1.4 (lines 10–16).[5] This is because combining items from $C_{i,i}$ and $C_{i,j}$ ($i \leq j$) may result in adding new items back into $C_{i,j}$, which must be allowed to combine with their progenitors in $C_{i,i}$ again. The usual duplicate check ensures that we will terminate with the same time bounds as before, but managing this incestuous computation requires a more general agenda-based control mechanism (Kay, 1986), whose weighted case is shown in Figure 1.5e.[6]

Analysis. The analysis is essentially the same as for polysemy (§4.2), i.e., $O(n^3g^3t^2)$ time, or $O(n^3g^3t)$ if we use the speedup of §3.4. The priority queue in Figure 1.5e introduces an extra factor of $\log |Agenda| = O(\log ngt)$. An ordinary FIFO or LIFO queue can be substituted in the unweighted case or if there are no cycles of the form discussed.[7]

However, g now bounds the number of possible *triples* (i, w, r) compatible with a position i in the input Ω. Notice that as with ℓ_w and r_w, there is no penalty for the number of arcs in R, i.e. the sizes of the vocabularies V, X.

Is g small? The intuition is that most transductions of interest give a small bound g, since they are locally "almost" invertible: they are constrained by the surface string Ω to only consider a few possible underlying words and states at each position i. For example, a transducer to handle polysemy (map senses

onto words) allows only a few underlying senses w per surface word W_i, and it needs only one state r.

But alas, the algorithm so far does not respect these constraints. Consider the SEED rule in Figure 1.5a: w (though not w') is allowed to take any value in V regardless of the input, and r, r' are barely more constrained. So the parser would allow many unnecessary triples and run very slowly. We now fix it to reclaim the intuition above.

Restoring Efficiency. We wish to constrain the (i, w, r) triples actually considered by the parser, by considering W_i and more generally the broader context provided by the entire input Ω. A triple (i, w, r) should never be considered unless it is consistent with some transduction that could have produced Ω.

We introduce two new kinds of items that let us check this consistency. The rules in Figure 1.5 derive the "forward item" $\xrightarrow{i,w} r$ iff R can terminally transduce αw (for some α) to $W_{1,i}$ on a subpath from $I(R)$ to r. They derive the "backward item" $r \xrightarrow{i}$ iff R can transduce some β to $W_{i+1,n}$ on a subpath from r to a final state. Figure 1.5d modifies the SEED rule to require such items as antecedents, which is all we need.

Remark. The new antecedents are used only as a filter. In parsing, they contribute no weight or detail to the analyses produced by the revised rule SEED. However, their weights might be used to improve parsing efficiency. Work by (Caraballo and Charniak, 1998) on best-first parsing suggests that the total weight of the three items

$$\xrightarrow{i,w} r \qquad i, w, r \quad \square \quad j, w', r' \qquad r' \xrightarrow{j}$$

may be a good heuristic measure of the viability of the middle item (representing a type of span) in the context of the rest of the sentence. (Notice that the middle item cannot be derived at all unless the other two also can.)

5. RELATION TO OTHER FORMALISMS

The bilexical grammar formalism presented here is flexible enough to capture a variety of grammar formalisms and probability models. On the other hand, as discussed in §5.6, it does not achieve the (possibly unwarranted) power of certain other bilexical formalisms.

5.1 MONOLEXICAL DEPENDENCY GRAMMAR

Lexicalized Dependency Grammar. It is straightforward to encode dependency grammars such as those of (Gaifman, 1965). We focus here on the case that (Milward, 1994) calls Lexicalized Dependency Grammar (LDG). Milward

demonstrates a parser for this case that requires $O(n^3g^3t^3)$ time and $O(n^2g^2t^2)$ space, using a left-to-right algorithm that maintains its state as an acyclic directed graph. Here t is taken to be the maximum number of dependents on a word.

LDG is defined to be only *monolexical*. Each word sense entry in the lexicon is for a word tagged with the type of phrase it projects. An entry for *helped*/S, which appears as head of the sentence *Nurses helped John wash*, may specify that it wants a left dependent sequence of the form w_1/N and a right dependent sequence of the form w_2/N, w_3/V. However, under LDG it cannot constrain the lexical content of w_1, w_2, or w_3, either discretely or probabilistically.[8]

By encoding a monolexical LDG as a bilexical grammar, and applying the algorithm of this chapter, we can reduce parsing time and space by factors of t^2 and t, respectively. The encoding is straightforward. To capture the preferences for *helped*/S as above, we define $\ell_{helped/S}$ to be a two-state automaton that accepts exactly the set of nouns, and $r_{helped/S}$ to be a three-state automaton that accepts exactly those word sequences of the form (noun, verb).

Obviously, $\ell_{helped/S}$ includes a great many arcs—one arc for every noun in V. This does not however affect parsing performance, which depends only on the number of *states* in the automaton.

Optional and Iterated Dependents. The use of automata to specify dependents is similar to the idea of allowing regular expressions in CFG rules, e.g., NP $\rightarrow$ (Det) Adj* N (Woods, 1969). It makes the bilexical grammar above considerably more flexible than the LDG that it encodes. In the example above, $r_{helped/S}$ can be trivially modified so that the dependent verb is optional (*Nurses helped John*). LDG can accomplish this only by adding a new lexical sense of *helped/S*, increasing the polysemy term g.

Similarly, under a bilexical grammar, $\ell_{nurses/N}$ can be specified to accept dependent sequences of the form (adj, adj, adj, ... adj, (det)). Then *nurses* may be expanded into *weary Belgian nurses*. Unbounded iteration of this sort is not possible in LDG, where each word sense has a fixed number of dependents. In LDG, as in categorial grammars, *weary Belgian nurses* would have to be headed by the adjunct *weary*. Thus, even if LDG were sensitive to bilexicalized dependencies, it would not recognize *nurses*→*helped* as such a dependency in *weary Belgian nurses helped John*. (It would see *weary*→*helped* instead.)

5.2 BILEXICAL DEPENDENCY GRAMMAR

In the example of §5.1, we may arbitrarily weight the individual noun arcs of the ℓ_{helped} automaton, according to how appropriate those nouns are as subjects of *helped*. (In the unweighted case, we might choose to rule out inanimate subjects altogether, by removing their arcs or assigning them the weight $-\infty$.)

This turns the grammar from monolexical to bilexical, without affecting the cubic-time cost of the parsing algorithm of §3.3.

5.3 TEMPLATE MATCHING

(Becker, 1975) argues that much naturally-occurring language is generated by stringing together fixed phrases and templates. To the bilexical construction of §5.2, one may add handling for special phrases. Consider the idioms (a) *run scared*, (b) *run circles [around* NP*]*, and (c) *run* NP *[into the ground]*. (a), like most idioms, is only bilexical, so it may be captured 'for free': simply increase the weight of the *scared* arc in $r_{run/V}$. But because (b) and (c) are trilexical, they require augmentation to the grammar, possibly increasing t and g. (b) requires a special state to be added to $r_{run/V}$, so that the dependent sequence (*circles*, *around*) may be recognized and weighted heavily. (c) requires a specialized lexical entry for *into*; this sense is a preferred dependent of *run* and has *ground* as a preferred dependent.

5.4 PROBABILISTIC BILEXICAL MODELS

(Eisner, 1996a) compares several distinct probability models for dependency grammar. Each model simultaneously evaluates the part-of-speech tags and the dependencies in a given dependency parse tree. Given an untagged input sentence, the goal is to find the tagged dependency parse tree with highest probability under the model.

Each of these models can be accomodated to the bilexical parsing framework, allowing a cubic-time solution. In each case, V is a set of part-of-speech-tagged words. Each weighted automaton ℓ_w or r_w is defined so that it accepts any dependent sequence in V^*—but the automaton has 8 states, arranged so that the weight of a given dependent w' (or the probability of halting) depends on the major part-of-speech category of the previous dependent.[9] Thus, any arc that reads a noun (say) terminates in the Noun state. The w'-reading arc *leaving* the Noun state may be weighted differently from the w'-reading arcs from other states; so the word w' may be more or less likely as a child of w according to whether its preceding sister was a noun.

As sketched in (Eisner, 1996b), each of Eisner's probability models is implemented as a particular scheme for weighting these automaton. For example, model C regards ℓ_w and r_w as Markov processes, where each state specifies a probability distribution over its exit options, namely, its outgoing arcs and the option of halting. The weight of an arc or a final state is then the log of its probability. Thus if $r_{helped/V}$ includes an arc labeled with $bathe/V$ and this arc is leaving the Noun state, then the arc weight is (an estimate of)

$$\log \Pr(\text{next right dependent is } bathe/V \mid \text{parent is } helped/V \text{ and previous right dependent was a noun})$$

The weight of a dependency parse tree under this probability model is a sum of such factors, which means that it estimates $\log \Pr(\text{dependency links \& input words})$ according to a generative model. By contrast, model D estimates $\log \Pr(\text{dependency links} \mid \text{input words})$, using arc weights that are roughly of the form

$$\log \Pr(bathe/V \text{ is a right dep. of } helped/V \mid \text{both words appear in sentence and prev. right dep. was a noun})$$

which is similar to the probability model of (Collins, 1996). Thus, different probability models are simply different weighting schemes within our framework. Some of the models use the trigram weighting approach of §4.3.

5.5 BILEXICAL PHRASE-STRUCTURE GRAMMAR

Nonterminal Categories as Sense Distinctions. In some situations, conventional phrase-structure trees appear preferable to dependency trees. (Collins, 1997) observes that since VP and S are both verb-headed, the dependency grammars of §5.4 would falsely expect them to appear in the same environments. (The expectation is false because *continue* subcategorizes for VP only.) Phrase-structure trees address the problem by subcategorizing for phrases that are labeled with nonterminals like VP and S.

Within the present formalism, the solution is to distinguish multiple senses (§4.2) for each word, one for each of its possible maximal projections. Then *help/VP*$_{\text{inf}}$ and *help/S* are separate senses: they take different dependents (yielding *to help John* vs. *nurses help John*), and only the former is an appropriate dependent of *continue*.

Unflattening the Dependency Structure. A second potential advantage of phrase-structure trees is that they are more articulated than dependency trees. In a (headed) phrase-structure tree, a word's dependents may attach to it at different levels (with different nonterminal labels), providing an obliqueness order on the dependents. Obliqueness is of semantic interest; it is also exploited by (Wu, 1995), whose statistical translation model preserves the topology (ID but not LP) of binary-branching parses.

For the most part, it is possible to recover this kind of structure under the present formalism. A scheme can be defined for converting dependency parse trees to labeled, binary-branching phrase-structure trees. Then one can use the fast bilexical parsing algorithm of §3.3 to generate the highest-weighted dependency tree, and then convert that tree to a phrase-structure tree, as shown in Figure 1.6.

For concreteness, we sketch how such a scheme might be defined. First label the states of all automata ℓ_w, r_w with appropriate nonterminals. For example, $r_{help/S}$ might start in state V; it transitions to state VP after reading its object,

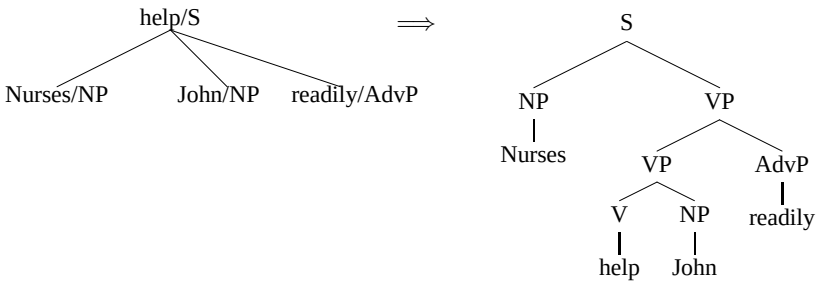


Figure 1.6 Unflattening a dependency tree when the word senses and automaton states bear nonterminal labels.

John/NP; and it loops back to VP when reading an adjunct such as *readily/AdvP*. Now, given a dependency tree for *Nurses help John readily*, we can reconstruct the sequence V, VP, VP of states encountered by $r_{help/S}$ as it reads *help*'s right children, and thereby associate a nonterminal attachment level with each child.

To produce the full phrase-structure tree, we must also decide on an obliqueness order for the children. Since this amounts to an order for the nodes at which the children attach, one approach is to derive it from a preferred total ordering on node types, according to which, say, right-branching VP nodes should always be lower than left-branching S nodes. We attach the children one at a time, referring to the ordering whenever we have a choice between attaching the next left child and the next right child.

This kind of scheme is adequate for most linguistic purposes. (For example, together with polysemy (§4.2) it can be used to encode the Treebank grammars of (Charniak, 1995).) It is interesting to compare it to (Collins, 1996), who maps phrase-structure trees to dependency trees whose edges are labeled with triples of nonterminals. In that paper Collins defines the probability of a phrase-structure tree to be the probability of its corresponding dependency tree. However, since his map is neither 'into' nor 'onto,' this does not quite yield a probability distribution over phrase-structure trees; nor can he simply find the best dependency tree and convert it to a phrase-structure tree as we do here, since the best dependency tree may correspond to 0 or 2 phrase-structure trees.

Neither the present scheme nor that of (Collins, 1996) can produce arbitrary phrase-structure trees. In particular, they cannot produce trees in which several adverbs alternately left-adjoin and right-adjoin to a given VP. We now consider the more powerful class of head-automaton grammars and bilexical context-free grammars, which *can* describe such trees.

5.6 HEAD AUTOMATA

Weighted bilexical grammars are essentially a special case of head-automaton grammars (Alshawi, 1996). As noted in the introduction, HAGs are bilexical in spirit. However, the left and right dependents of a word w are accepted not separately, by automata ℓ_w and r_w, but in interleaved fashion by a single weighted automaton, d_w. d_w assigns weight to strings over the alphabet $V \times \{\leftarrow, \rightarrow\}$; each such string is an interleaving of lists of left and right dependents from V.

Head automata, as well as (Collins, 1997), can model the case that §5.5 cannot: where left and right dependents are arbitrarily interleaved. (Alshawi, 1996) points out that this makes head automata fairly powerful. A head automaton corresponding to the regular expression $((a, \leftarrow)(b, \rightarrow))^*$ requires its word to have an equal number of left and right dependents, i.e,. $a^n w b^n$. (Bilexical or dependency grammars are context-free in power, so they can also generate $\{a^n w b^n : n \geq 0\}$—but only with a structure where the a's and b's depend bilexically on each other, not on w. Thus, they allow only the usual linguistic analysis of the doubly-center-embedded sentence *Rats cats children frequently mistreat chase squeak*.)

For syntactic description, the added generative power of head automata is probably unnecessary. (Linguistically plausible interactions among left and right subcat frames, such as fronting, can be captured in bilexical grammars simply via multiple word senses.)

Head automaton grammars and an equivalent bilexical CFG-style formalism are discussed further in (Eisner and Satta, 1999), where it is shown that they can be parsed in time $O(n^4 g^2 t^2)$.

5.7 LINK GRAMMARS

There is a strong connection between the algorithm of this chapter and the $O(n^3)$ link grammar parser of (Sleator and Temperley, 1993). As Alon Lavie (p.c.) has pointed out, both algorithms use essentially the same decomposition into what are here called spans. Sleator and Temperley's presentation (as a top-down memoizing algorithm) is rather different, as is the parse scoring model introduced by (Lafferty et al., 1992). (Link grammars were unknown to this author when he developed and implemented the present algorithm in 1994.)

This section makes the connection explicit. It gives a brief (and attractive) definition of link grammars and shows how a minimal variant of the present algorithm suffices to parse them. As before, our algorithm allows an arbitrary weighting model (§4.1) and can be extended to parse the composition of a link grammar and a finite-state transducer (§4.4).

Formalism. A link grammar may be specified exactly as the bilexical grammars of §2. are. A link grammar parse of $\Omega = W_1 W_2 \ldots W_n$, called a **linkage**,

is a connected undirected graph whose vertices $\{1, 2, \ldots n+1\}$ are respectively labeled with $w_1 \in W_1, w_2 \in W_2, \ldots w_n \in W_n, w_{n+1} = \text{ROOT}$, and whose edges do not 'cross,' i.e., edges i–k and j–ℓ do not both exist for any $i < j < k < \ell$. The linkage is grammatical iff for each vertex i, ℓ_{w_i} accepts the sequence of words $\langle w_j : j < i, i\text{–}j \text{ is an edge}\rangle$ (ordered by decreasing j), and r_{w_i} accepts the sequence of words $\langle w_j : j > i, i\text{–}j \text{ is an edge}\rangle$ (ordered by increasing j).

Traditionally, the edges of a linkage are labeled with named grammatical relations. In this case, ℓ_{w_i} should accept the sequence of pairs $\langle (w_j, R) : j < i, i\text{–}j \text{ is an edge labeled by } R\rangle$, and similarly for r_{w_i}.

Discussion. The above formalism improves slightly on (Sleator and Temperley, 1993) by allowing arbitrary DFAs rather than just straight-line automata (cf. §5.1). This makes the formalism more expressive, so that it is typically possible to write grammars with a lower polysemy factor g. In addition, any weights or probabilities are sensitive to the underlying word senses w_i (known in link grammar as **disjuncts**), not merely the surface graphemes W_i.

Allowing finite-state post-processing as in §4.4 also makes the formalism more expressive. It allows a modular approach to writing grammars: the link grammar handles dependencies (topology-local phenemona) while the transducer handles string-local phenomena.

Modifying the Algorithm. Linkages have a less restricted form than dependency trees. Both are connected graphs without crossing edges, but only dependency trees disallow cycles or distinguish parents from children. The algorithm of Figure 1.3 therefore had to take extra pains to ensure that each word has a unique directed path to ROOT. It can be simplified for the link grammar case, where we only need to ensure connectedness. In place of the bits b_1 and b_2, the signature of an analysis of $w_{i,j}$ should include a single bit indicating whether the analysis is a connected graph; if not, it has two connected components. The input to ACCEPT and at least one input to COMBINE must be connected. (As for output, obviously SEED's output is not connected, OPT-LINK's is, and COMBINE or SEAL's output is connected iff all its inputs are.) To prevent linkages from becoming multigraphs, each item needs an extra bit indicating whether it is the output of OPT-LINK; if so, it may not be input to OPT-LINK again.

Figure 1.3 (or Figure 1.5) needs one more change to become an algorithm for link grammars. There should be only one OPT-LINK rule, which should advance the state q_1 of r_{w_i} to some state q_1' by reading w_j (like OPT-LINK-L), and *simultaneously* advance the state q_2 of ℓ_{w_j} to some state q_2' by reading w_i (like OPT-LINK-R). (Or if edges are labeled, there must be a named relation R such that r_{w_i} reads (w_j, R) and ℓ_{w_j} reads (w_i, R).) This is because link

grammar's links are not directional: the linked words w_i and w_j stand in a symmetric relation wherein they must accept each other.

Analysis. The resulting link grammar parser runs in time $O(n^3g^3t^2)$; so does the obvious generalization of (Sleator and Temperley, 1993) to our automaton-based formalism. A minor point is that t is measured differently in the two algorithms, since the automata ℓ_w, r_w used in the Sleator-Temperley-style top-down algorithm must be the reverse of those used in the above bottom-up algorithm. (The minimal DFAs accepting a language L and its reversal L^R may have exponentially different sizes t.)

The improvement of §3.4 to $O(n^3g^3t)$ is not available for link grammars. Nor is the improvement of (Eisner and Satta, 1999) to $O(n^3g^2t)$, which uses a different decomposition that relies on acyclicity of the dependency graph.

5.8 LEXICALIZED TREE-ADJOINING GRAMMARS

The formalisms discussed in this chapter have been essentially context-free. The kind of $O(n^3)$ or $O(n^4)$ algorithms we have seen here cannot be expected for the more powerful class of mildly context-sensitive grammars (Joshi et al., 1991), where the best known parsing algorithms are $O(n^6)$ even for *non*-lexicalized cases. However, it is worth remarking that similar problems and solutions apply when bilexical preferences are added. In particular, Lexicalized Tree-Adjoining Grammar (Schabes et al., 1988) is actually bilexical, since each tree contains a lexical item and may select for other trees that substitute or adjoin into it. (Eisner and Satta, 2000) show that standard TAG parsing essentially takes $O(n^8)$ in this case, but can be sped up to $O(n^7)$.

6. CONCLUSIONS

Following recent trends in probabilistic parsing, this chapter has introduced a new grammar formalism, weighted bilexical grammars, in which individual lexical items can have idiosyncratic selectional influences on each other.

The new formalism is derived from dependency grammar. It can also be used to model other bilexical approaches, including a variety of phrase-structure grammars and (with minor modifications) all link grammars. Its scoring approach is compatible with a wide variety of probability models.

The obvious parsing algorithm for bilexical grammars (used by most authors) takes time $O(n^5g^2t)$. A new method is exhibited that takes time $O(n^3g^3t)$. An extension parses sentences that have been "corrupted" by a rational transduction.

The simplified $O(n^3g^3t^2)$ variant of §3.3 was originally sketched in (Eisner, 1996b) and presented (though without benefit of Figure 1.3) in (Eisner, 1997). It has been used successfully in a large parsing experiment (Eisner, 1996a).

The reader may wish to know that more recently, (Eisner and Satta, 1999) found an alternative algorithm that combines half-constituents rather than spans. It has the same space requirements, and the asymptotically faster runtime of $O(n^3g^2t)$—achieving the same cubic time on the input length but with a grammar factor as low as that of the naive n^5 algorithm.

While the algorithm presented in this chapter is not as fast asymptotically as that one, there are nonetheless a few reasons to consider using it:

- It is perhaps simpler to implement, as the chart contains not four types of subparse but only one.[10]

- With minor modifications (§5.7), the same implementation can be used for link grammar parsing. This does not seem to be true of the faster algorithm.

- In some circumstances, it may run faster despite the increased grammar constant. This depends on the grammar (i.e., the values of g and t) and other constants in the implementation.

- Using probabilities or a hard grammar to prune the chart can significantly affect average-case behavior. For example, in one unpublished experiment on Penn Treebank/*Wall Street Journal* text (reported by the author at ACL '99), probabilistic pruning closed the gap between the $O(n^3g^3t^2)$ and $O(n^3g^2t)$ algorithms. (Both still substantially outperformed the pruned $O(n^5)$ algorithm.)

- With the improvement presented in §3.4, the asymptotic penalty of the span-based approach presented here is reduced to only $O(g)$.

Thus, while (Eisner and Satta, 1999) is the safer choice overall, the relative performance of the two algorithms in practice may depend on various factors.

One might also speculate on algorithms for related problems. For example, the g^3 factor in the present algorithm (compared to Eisner and Satta's g^2) reflects the fact that the parser sometimes considers three words at once. In principle this could be exploited. The probability of a dependency link could be conditioned on all three words or their senses, yielding a 'trilexical' grammar. (Lafferty et al., 1992) use precisely such a probability model in their related $O(n^3)$ algorithm for parsing link grammars, although it is not clear how relevant their third word is to the probability of the link (Eisner, 1996b).

Acknowledgments

I am grateful to Michael Collins, Joshua Goodman, and Alon Lavie for useful discussion of this work.

Notes

1. Actually, (Lafferty et al., 1992) is formulated as a *trilexical* model, though the influence of the third word could be ignored: see §6..

2. Having unified an item with the left input of an inference rule, such as COMBINE in Figure 1.3, the parser must enumerate all items that can then be unified with the right input.

3. In the sense of the dotted rules of (Earley, 1970).

4. Notice that our assumption about the form of arc labels, above, guarantees that any span of T will be transduced to some substring of Ω by an an exact subpath of P. Without that assumption, the span might begin in the middle of some arc of P.

5. Cycles that transduce ϵ to ϵ would create a similar problem for the rules of Figure 1.5b, but R can always be transformed so as to eliminate such cycles.

6. We assume that the output of a rule is no heavier than any of its inputs, so that additional trips around a derivational cycle cannot increase weight unboundedly. (E.g., all rule weights are log-probabilities and hence ≤ 0.) In this case the code can be shown correct: it pops items from the agenda only after their highest-weighted (Viterbi) derivations are found, and never puts them back on the agenda.

The algorithm is actually a generalization to hypergraphs of the single-source shortest-paths algorithm of (Dijkstra, 1959). In a hypergraph such as the parse forest, each parent of a vertex (item) is a *set* of vertices (antecedents). Our single source is taken to be the empty antecendent set. Note that finding the *total* weight of all derivations would be much harder than finding the maximum, in the presence of cycles (Stolcke, 1995; Goodman, 1998).

7. The time required for the agenda-based algorithm is proportional to the number of rule instances used in the derivation forest. The space is proportional to the number of items derived.

8. What would happen if we tried to represent bilexical dependencies in such a grammar? In order to restrict w_2 to appropriate objects of *helped*/S, the grammar would need a new nonterminal symbol, $\mathrm{N}_{\mathrm{helpable}}$. All nouns in this class would then need additional lexical entries to indicate that they are possible heads of $\mathrm{N}_{\mathrm{helpable}}$. The proliferation of such entries would drive g up to $|V|$ in Milward's algorithm, resulting in $O(n^3|V|^3t^3)$ time (or by ignoring rules that do not refer to lexical items in the input sentence, $O(n^6t^3)$).

9. The eight states are START, Noun, Verb, Noun Modifier, Adverb, Prep, Wh-word, and Punctuation.

10. On the other hand, for indexing purposes it is helpful to partition this type into at least two subtypes: see the two charts of Figure 1.4.

References

Alshawi, H. (1996). Head automata and bilingual tiling: Translation with minimal representations. In *Proceedings of the 34th ACL*, pages 167–176, Santa Cruz, CA.

Becker, J. D. (1975). The phrasal lexicon. Report 3081 (AI Report No. 28), Bolt, Beranek, and Newman.

Caraballo, S. A. and Charniak, E. (1998). New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*.

Charniak, E. (1995). Parsing with context-free grammars and word statistics. Technical Report CS-95-28, Department of Computer Science, Brown University, Providence, RI.

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 598–603, Menlo Park. AAAI Press/MIT Press.

Church, K. W. (1988). A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the 2nd Conf. on Applied NLP*, pages 136–148, Austin, TX.

Collins, M. J. (1996). A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th ACL*, pages 184–191, Santa Cruz, July.

Collins, M. J. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th ACL and 8th European ACL*, pages 16–23, Madrid, July.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Eisner, J. (1996a). An empirical comparison of probability models for dependency grammar. Technical Report IRCS-96-11, Institute for Research in Cognitive Science, Univ. of Pennsylvania.

Eisner, J. (1996b). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, Copenhagen.

Eisner, J. (1997). Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the 1997 International Workshop on Parsing Technologies*, pages 54–65, MIT, Cambridge, MA.

Eisner, J. and Satta, G. (1999). Efficient parsing for bilexical context-free grammars and head-automaton grammars. In *Proceedings of the 37th ACL*, pages 457–464, University of Maryland.

Eisner, J. and Satta, G. (2000). A faster parsing algorithm for lexicalized tree-adjoining grammars. In *Proceedings of the 5th Workshop on Tree-Adjoining Grammars and Related Formalisms (TAG+5)*, Paris.

Gaifman, H. (1965). Dependency systems and phrase structure systems. *Information and Control*, 8:304–337.

Goodman, J. (1997). Probabilistic feature grammars. In *Proceedings of the 1997 International Workshop on Parsing Technologies*, pages 89–100, MIT, Cambridge, MA.

Goodman, J. (1998). *Parsing Inside-Out*. PhD thesis, Harvard University.

Graham, S. L., Harrison, M. A., and Ruzzo, W. L. (1980). An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2(3):415–463.

Joshi, A. K., Vijay-Shanker, K., and Weir, D. (1991). The convergence of mildly context-sensitive grammar formalisms. In Sells, P., Shieber, S. M., and Wasow, T., editors, *Foundational Issues in Naural Language Processing*, chapter 2, pages 31–81. MIT Press.

Kaplan, R. M. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Kay, M. (1986). Algorithm schemata and data structures in syntactic processing. In Grosz, B. J., Sparck Jones, K., and Webber, B. L., editors, *Natural Language Processing*, pages 35–70. Kaufmann, Los Altos, CA.

Koskenniemi, K. (1983). Two-level morphology: A general computational model for word-form recognition and production. Publication 11, Department of General Linguistics, University of Helsinki.

Lafferty, J., Sleator, D., and Temperley, D. (1992). Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, pages 89–97, Cambridge, MA.

McAllester, D. (1999). On the complexity analysis of static analyses. In *Proceedings of the 6th International Static Analysis Symposium*, Venezia, Italy.

Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press.

Milward, D. (1994). Dynamic dependency grammar. *Linguistics and Philosophy*, 17:561–605.

Mohri, M., Pereira, F., and Riley, M. (1996). Weighted automata in text and speech processing. In *Workshop on Extended Finite-State Models of Language (ECAI-96)*, pages 46–50, Budapest.

Pollard, C. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press and Stanford: CSLI Publications, Chicago.

Resnik, P. (1993). *Selection and Information: A Class-Based Approach to Lexical Relationships*. PhD thesis, University of Pennsylvania. Technical Report IRCS-93-42, November.

Schabes, Y., Abeillé, A., and Joshi, A. (1988). Parsing strategies with 'lexicalized' grammars: Application to Tree Adjoining Grammars. In *Proceedings of COLING-88*, pages 578–583, Budapest.

Sleator, D. and Temperley, D. (1993). Parsing English with a link grammar. In *Proceedings of the 3rd International Workshop on Parsing Technologies*, pages 277–291.

Stolcke, A. (1995). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.

Woods, W. A. (1969). Augmented transition networks for natural language analysis. Report CS-1, Harvard Computation Laboratory, Harvard University, Cambridge, MA.

Wu, D. (1995). An algorithm for simultaneously bracketing parallel texts by aligning words. In *Proceedings of the 33rd ACL*, pages 244–251, MIT.