EVALUATING GRAMMAR FORMALISMS FOR APPLICATIONS TO NATURAL LANGUAGE PROCESSING AND BIOLOGICAL SEQUENCE ANALYSIS

David Chiang

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2004

JMJ

Acknowledgments

I would like to thank my adviser Aravind Joshi, a wise teacher he is, and Ken Dill at UCSF for his inspiring guidance. Thanks also to my dissertation committee, Fernando Pereira, Jean Gallier, Sampath Kannan, and Giorgio Satta for their careful reading and feedback; and other faculty connected with Penn whose advice has been valuable to me: Mark Steedman, Mitch Marcus, Martha Palmer, and David Searls.

Among the grad students and postdocs I have interacted with at Penn, I am particularly indebted to Dan Bikel, Mark Dras, and William Schuler. My collaborations with them and countless discussions taught me half of what I know about computational linguistics. The other half no doubt comes from the rest, including Mike Collins, Dan Gildea, Julia Hockenmaier, Seth Kulick, Alexandra Kinyon, Tom Morton, Anoop Sarkar, and Fei Xia.

I thank Hoa Trang Dang and my sister Lillian for their loving support, and my parents, who have given me more than I could possibly thank them for.

The research described in this thesis was supported in part by the following grants: ARO DAAG5591-0228, NSF SBR-89-20230, and NSF ITR EIA-02-05456.

ABSTRACT

EVALUATING GRAMMAR FORMALISMS FOR APPLICATIONS
TO NATURAL LANGUAGE PROCESSING
AND BIOLOGICAL SEQUENCE ANALYSIS

David Chiang

Supervisor: Aravind K. Joshi

Grammars are gaining importance in statistical natural language processing and computational biology as a means of encoding theories and structuring algorithms. But one serious obstacle to applications of grammars is that formal language theory traditionally classifies grammars according to their weak generative capacity (WGC)—what sets of strings they generate—and tends to ignore strong generative capacity (SGC)—what sets of structural descriptions they generate—even though the latter is more relevant to applications.

This dissertation develops and demonstrates, for the first time, a framework for carrying out rigorous comparisons of grammar formalisms in terms of their usefulness for applications. We do so by adopting Miller's view of SGC as pertaining not directly to structural descriptions but their interpretations in particular domains; and, following Joshi et al., by appropriately constraining the grammars and interpretations we consider. We then consider three areas of application.

The first area is statistical parsing. We find that, in this domain, attempts to increase the SGC of a formalism can often be compiled back into the simpler formalism, gaining nothing. But this suggests a new view of current parsing models as compiled versions of grammars from richer formalisms. We discuss the implications of this view and its implementation in a probabilistic tree-adjoining grammar model, with experimental results on English and Chinese.

For our other two applications, by contrast, we can readily increase the SGC of a formalism without increasing its computational complexity. For natural language translation, we discuss the formal, linguistic, and computational properties of a formalism that is more powerful than those

currently proposed for statistical machine translation systems.

Finally, we explore the application of formal grammars to modeling secondary/tertiary structures of biological sequences. We show how additional SGC can be used to extend models to take more complex structures into account, paying special attention to the technique of intersection, which has drawn comparatively little attention in computational linguistics.

These results should pave the way for theoretical research to pursue results that are more directed towards applications, and for practical research to explore the use of advanced grammar formalisms more easily.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> Then the three young men of the bodyguard, who kept guard over the person of the king, said to one another, Let each of us state what one thing is strongest; and to the one whose statement seems wisest, King Darius will give rich gifts and great honors of victory… The first wrote, Wine is strongest. The second wrote, The king is strongest. The third wrote, Women are strongest, but above all things truth is victor.
>
> —*1 Esdras* 3.4–5, 10–12

> One should realize… that if we consider these four, namely wine, the king, woman and truth, in themselves they are not comparable because they do not belong to the same genus. Nevertheless, if they are considered in relation to some effect, they coincide in one aspect, and so can be compared with each other.
>
> —St. Thomas Aquinas, *Quaestiones quodlibetales*, XII, q. 14, a. 1

Formal grammars, first developed as specifications of linguistic theories and programming languages, have found a rich variety of applications in computer science, especially in natural language processing and, more recently, biological sequence analysis. A question that naturally arises in such applications is: What formalism should grammars be expressed in? What makes one formalism better than another? For our purposes, we may view a grammar formalism simply as the set of all grammars it can express; the larger this set, the better. But this only reduces the question to another question: What makes one grammar equivalent to another?

Formal language theory has traditionally given two ways of answering this question, namely, *weak generative capacity* (WGC) and *strong generative capacity* (SGC). The WGC of a grammar

is the set of *strings* it generates, and its SGC is the set of *structural descriptions* it assigns to them. The WGC of a formalism is then the set of WGCs of its grammars, and its SGC is the set of SGCs of its grammars. Occasionally one finds the term "strong generative capacity" misapplied to the set of phrase-structure trees a grammar generates (which we will refer to as its *tree generative capacity*). But a structural description may be any kind of structure (e.g., dependency structure, f-structure, derivation tree, proof tree) a grammar might assign to a string.

At the time that Chomsky introduced these terms, he observed that SGC is "by far the more interesting notion" (Chomsky and Miller, 1963, p. 297), but WGC is "the only area in which substantial results of a mathematical character have been achieved" (Chomsky, 1963, pp. 325–326). The reason SGC is more interesting is that it is via structural descriptions that the grammar interfaces with higher-level modules (i.e., semantics). But the paucity of results having to do with SGC is due, at least in part, to the difficulty of defining what it means for two structural descriptions to be equivalent, especially when they come from different formalisms. As Aquinas might say, they are not comparable because they do not belong to the same genus. Thus it can be extremely difficult to say what advantage one formalism has over another.

Forty years later, Chomsky's observation holds true. Formal language theory has produced many significant mathematical results, but continues to focus on WGC rather than SGC. Indeed, as more grammar formalisms are introduced, the more difficult it becomes to compare their structural descriptions and therefore their SGC. On the other hand, the advancement of computing technology has opened up new applications of formal grammars beyond generative syntax, and SGC is more relevant than WGC for these applications just as it was more relevant for generative syntax. But because results having to do with SGC are still lacking, the use of new grammar formalisms is too often justified by intuition or examples or not at all.

The problem of SGC has been addressed previously, if infrequently, in the context of formal linguistics (Kuroda, 1976; Kuroda, 1987; Miller, 1999), but hardly at all in the context of computer applications. This dissertation presents for the first time a comprehensive attempt to mathematically evaluate grammar formalisms for computer applications. Our thesis is that such evaluation is

made possible by appropriately constrained notions of SGC and is a helpful predictor of empirical performance.

1.1 Overview

Theoretical framework

In Sections 1.2 and 1.3 we set up the basic framework in which we will carry out our comparisons of grammar formalisms. Our notion of SGC is based on Miller's (1999). If the incommensurability of structural descriptions across formalisms is analogous to the incommensurability of "wine, the king, woman, and truth," then Miller's solution is analogous to Aquinas': he proposes not to compare structural descriptions directly, but to compare their denotations in particular *interpretation domains* (corresponding to Aquinas' "effects"). He proceeds to define various interpretation domains and demonstrates how different formalisms can be compared within those domains.

Our adaptation of Miller's notion of relativized SGC is more pragmatic. Whereas his interpretation domains capture linguistically significant information about structural descriptions, our interpretation domains are more application-driven, for example, the domain of probabilities. Moreover, because we are interested in the computation of interpretations, we constrain interpretation functions via constrained grammar formalisms. Following the approach taken in the literature on tree-adjoining grammars (TAGs), we consider grammars with well-defined domains of *locality*, allowing us to recursively define all interpretation functions in terms of *local interpretation functions*.

This framework allows us to systematically classify a wide range of grammar formalisms according to their power in various interpretation domains. Different interpretation domains classify formalisms differently: some will be coarser-grained, some will be finer-grained; it can even happen that one formalism is more powerful than another in one domain, but less powerful in another domain. We are especially interested in situations where we can "squeeze" SGC (Joshi, 2003) out

of a formalism while preserving its computational properties. We try to capture this intuition using the notion of a *cover* (Nijholt, 1980): a situation when one grammar is parsed using another grammar (the cover grammar) and therefore inherits its computational properties. Formalisms that increase SGC while preserving coverability are especially interesting. But formalisms that increase SGC without preserving coverability are interesting as well.

In this framework we will explore three areas of application: statistical natural language parsing, natural language translation, and modeling of biological macromolecules.

Statistical parsing

We first consider, in Chapter 3, the task of statistical parsing: computing the most likely structure (standardly, the most likely phrase-structure tree) of a given string. We discuss two types of statistical models: *history-based models*, which are easy to estimate, and *maximum-entropy models*, which are more general but more difficult to estimate. History-based models based on formal grammars are already well-understood; we discuss maximum-entropy models based on formal grammars and how to estimate them using an algorithm due to Miyao and Tsujii (2002).

Both types of model are subsumed by semiring-weighted grammars (Goodman, 1999); thus we can measure statistical-modeling power by SGC with respect to semiring-weighted structures. This notion of SGC turns out to classify formalisms rather coarsely: if a grammar G can be covered by (say) a CFG, then weights can also be assigned to the cover grammar to make it strongly equivalent to G with respect to weighted strings. In other words, we do not expect in general that formalisms which "squeeze" SGC out of CFG will provide any more statistical parsing models than weighted CFG does (although a more precise treatment of the question below leaves some room for exceptions).

But if the statistical-modeling power of these "squeezed" formalisms is already accessible within PCFG, then we can investigate how that power may already be utilized by existing PCFG models like those of Charniak (1997) or Collins (1997), which represented a breakthrough in statistical parsing. It turns out that the style of CFGs, called *lexicalized CFGs*, that these models use

is very similar to the cover CFGs from the above result. Thus, applying the construction in reverse to a lexicalized PCFG model yields a reinterpretation of lexicalized PCFG as a special kind of probabilistic TAG.

But TAG structural descriptions contain more information than the phrase-structure trees found in typical training corpora like the Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993). Under this interpretation, then, it becomes more clear that the purpose of the head-propagation rules used by lexicalized PCFG models is not simply to rearrange information in the training data, but to *reconstruct* information missing from the training data; and this information is not lexical, but *structural*.

We then describe our implementatation of this interpretation in a parsing model based on probabilistic tree-insertion grammar with sister-adjunction (TIG-SA). We reinterpret the head/argument rules from Magerman's SPATTER parser (Magerman, 1995) and Collins' parser (Collins, 1997) as a heuristic for reconstructing full structural descriptions from partial ones; we also explore a method related to the approach of Hwa (1998) which uses Expectation-Maximization to directly estimate the model defined over full structural descriptions on the partial structural descriptions in the training data. We present experimental results for both of these techniques, training our probabilistic TIG-SA model from the Penn Treebank (English) and the Penn Chinese Treebank. We find that our probabilistic TIG-SA model performs at roughly the same level as lexicalized PCFG models and explore some new directions of research that such a model opens up.

Natural language translation

The next application we discuss is translation (Chapter 4), for which the relevant notion of SGC is that of string pairs. In contrast to the classification above, this notion of SGC classifies formalisms quite finely. We analyze several synchronous formalisms proposed in the literature and add a new one, synchronous regular-form TAG (Dras, 1999; Chiang, Schuler, and Dras, 2000; Chiang, 2002). Whereas these formalisms are all weakly equivalent and have the same statistical-modeling power, they all differ in their translation power, that is, their SGC with respect to string or tree pairs.

We observe that machine translation research is following statistical parsing in moving towards more complex structural descriptions. But we argue that because translation power classifies formalisms so much more finely than statistical-modeling power, it is much more important in translation research to target the right formalism. We discuss how synchronous RF-TAG's formal, linguistic, and computational properties make it potentially well-suited for use in translation systems.

Biological sequence analysis

Finally, in Chapter 5 we explore the use of formal grammars for biological sequence analysis. Proteins and RNAs are folded out of molecules which are chains of building blocks (amino acids and nucleotides) assembled in a sequence specified by genes. The task of biological sequence analysis is to relate genetic sequences to the folded structures they encode. It was Searls (1992) who first observed the similarity between biological sequence analysis and natural-language-syntactic analysis and proposed that the same techniques could be applied to both problems. We give a unified treatment of previous applications of formal grammars to this problem, highlighting in particular their shared assumption that grammatical locality corresponds to physical locality. This observation implies that the relevant notion of SGC for this problem is that of *linked* strings. Searls' original work was on CFG; we explore some old and new ways of employing formalisms with greater SGC than CFG to model more complex structures: α-helices, β-sheets, kissing hairpins, and pseudoknots.

Most grammatical approaches to biological sequence analysis rank structures using weights—usually probabilities or energies. We describe a more sophisticated use of weights, drawing on a model due to Chen and Dill (1995; 1998) which tries not only to predict structures of chain molecules but to give a full description of their statistical-mechanical properties. Their model is not explicitly grammatical, but we show that it can be more cleanly viewed as a weighted CFG.

In Chapter 6, we explore a family of approaches based on the technique of *intersection*—analyzing a string with two or more grammars and composing their structural descriptions. Intersection is not used much for natural language, probably because hierarchical structural descriptions do not compose easily, but is more promising for biological sequence analysis, because there is a well-defined way of composing structures. We show how our CFG version of Chen and Dill's model can be intersected with a finite-state automaton for α-helices, easily yielding a novel model for bundles of α-helices. We also discuss how simple literal movement grammars (Groenink, 1997) (similar to range concatenation grammars (Boullier, 2000)) might use their built-in intersection operation to efficiently model protein β-sheets.

1.2 Example: Dutch, DGC, and TAG

The controversy over the complexity of Dutch is a classic illustration of the kinds of issues involved in testing the adequacy of a grammar formalism, which will guide us as we develop the theoretical framework for our own comparisons of grammar formalisms. One early argument against the adequacy of context-free grammar for natural language was put forth by Huybregts (1976). He argued that Dutch sentences exhibiting *cross-serial dependencies* like the following:

(1.1) dat Jan Piet de kinderen zag helpen zwemmen
that Jan Piet the children saw help swim
that Jan saw Piet help the children swim

(where the first NP is the subject of the first verb, the second of the second, and so on) show that Dutch is like the copy language $\{ww\}$, which is non-context-free. Pullum and Gazdar (1982) correctly replied that the sequence of verbs is not a copy of the sequence of nouns; the two sequences only had to be the same length. Therefore Dutch, considered as a set of strings, cannot be shown in any formal way to be reducible to the copy language.

Bresnan et al. (1982) argued using traditional constituency arguments that the phrase-structure trees of sentences like (1.1) had to have a certain form, and then proved that CFG cannot generate such tree sets, concluding that CFG does not have enough "strong generative capacity" (in our

terminology, tree generative capacity) to capture this construction. However, because it relied on theory-internal assumptions to determine the desired trees, this argument was not compelling.

Finally Huybregts (1984) and Shieber (1985) independently observed that Swiss German allows a cross-serial word order as Dutch does but also has verbs which mark their objects with different cases.

(1.2) das mer d'chind em Hans es huus lönd hälfe aastriiche
that we the children-ACC Hans-DAT the house-ACC let help paint
that we let the children help Hans paint the house

(1.3) ∗das mer d'chind de Hans es huus lönd hälfe aastriiche
that we the children-ACC Hans-ACC the house-ACC let help paint
that we let the children help Hans paint the house

Therefore Swiss German can be reduced via homomorphisms and intersections to the copy language $\{ww \mid w \in a^*b^*\}$, which proves that Swiss German is not a context-free language. But this still does not settle the question for Dutch.

The most satisfying answer to the question of Dutch comes from the literature on tree-adjoining grammars and related formalisms. Just as CFGs generate strings by rewriting symbols as strings, *tree-adjoining grammars* (Joshi, Levy, and Takahashi, 1975; Joshi and Schabes, 1997) generate trees by rewriting nodes as trees.

For example, consider the TAG of Figure 1.1. The tree α is called an *initial tree*, analogous to the start symbol in CFG. The trees β_1 and β_2 are called *auxiliary trees*, analogous to productions in CFG. They differ from initial trees in that they have exactly one frontier node marked with an asterisk ($*$); this node is called the *foot node* and always has the same label as the root node. The path from the root node to the foot node of an auxiliary tree is called its *spine*.

The basic rewriting operation is called *adjunction*, in which a node is rewritten with the spine of an auxiliary tree β along with all its branches. The rewritten node and the root/foot of β must have the same label. For example, Figure 1.2 shows the result of rewriting the lower S node of α with β_1.

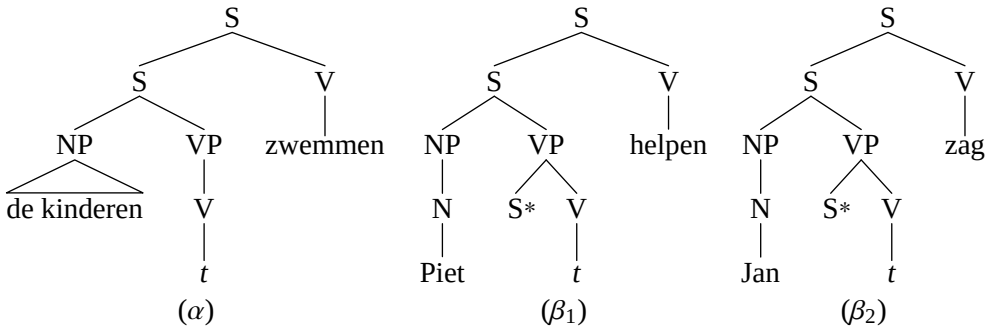


Figure 1.1: TAG for Dutch cross serial dependencies in sentence (1.1).

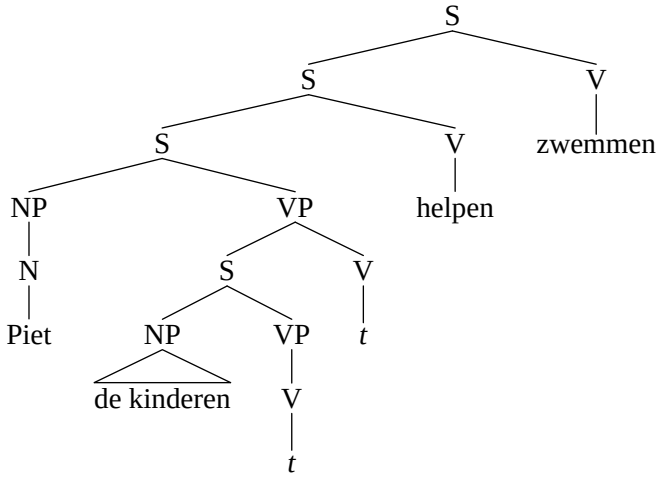


Figure 1.2: Example of adjunction. The tree β_1 has been adjoined at the lower S node of α.

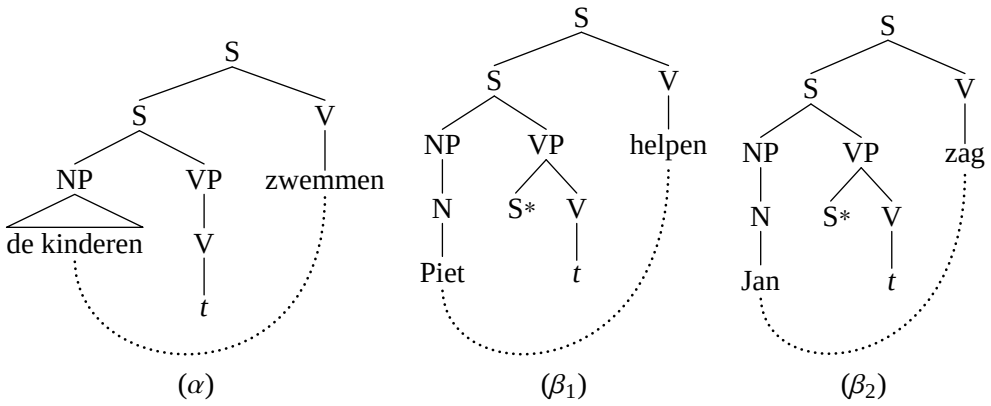


Figure 1.3: TAG with links for Dutch cross-serial dependencies in sentence (1.1).

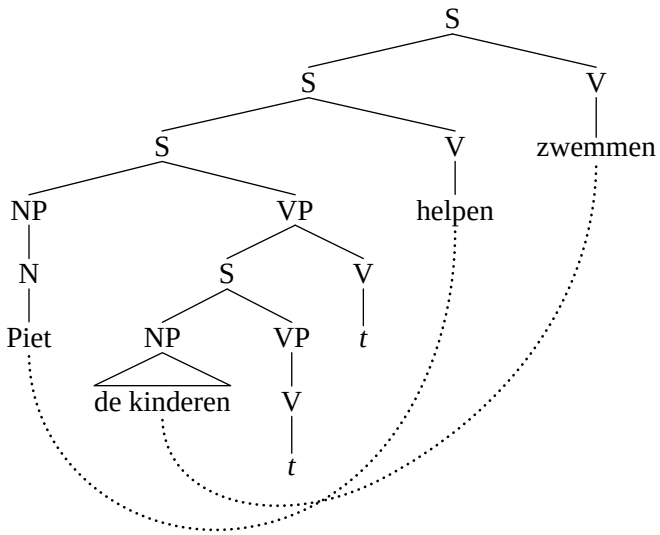


Figure 1.4: First step in derivation of cross-serial dependencies. The tree β_1 has been adjoined at the lower S node of α.

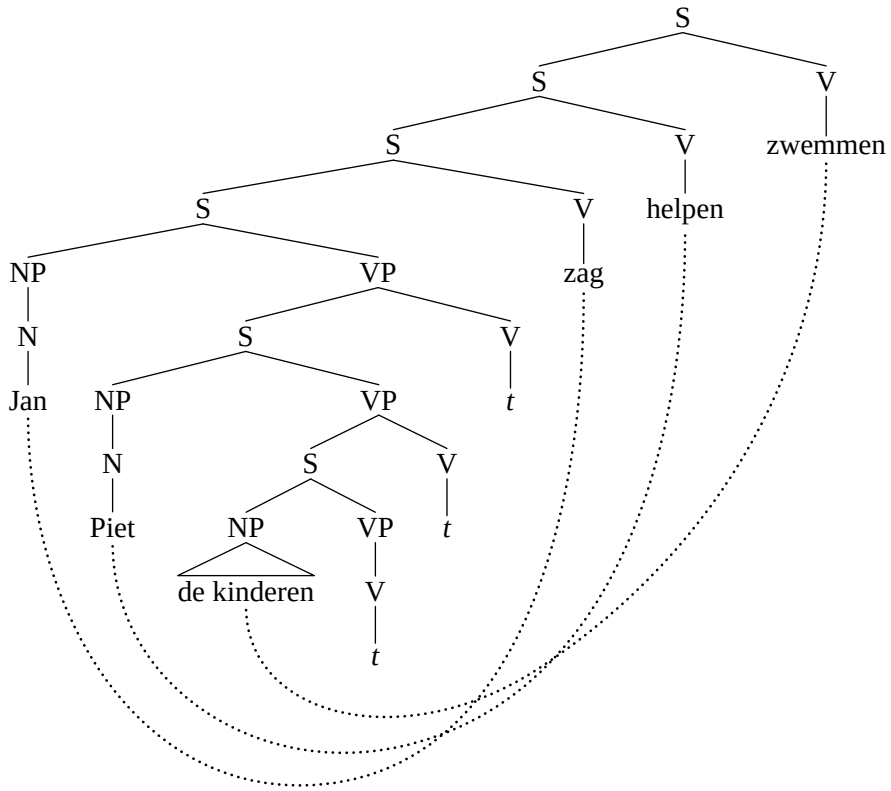
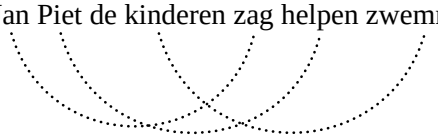


Figure 1.5: Second step in derivation of cross-serial dependencies. The tree β_2 has been adjoined at the third S node of Figure 1.4.

Joshi (1985) showed that a tree-adjoining grammar with *links* (see Figure 1.3; this particular analysis is due to Kroch and Santorini (1991)) could generate examples like (1.1) with the dependencies (which are much less controversial than the phrase structure) explicitly marked:

(1.4) dat Jan Piet de kinderen zag helpen zwemmen



Figures 1.4 and 1.5 show the derivation of the sentence, with the cross-serial dependencies as desired. This is possible with a TAG but not possible for any CFG. There are two key assumptions at work in this argument. First, grammars are not measured according to their WGC but according to their ability to generate strings with subject-verb dependencies explicitly marked with links. Second, CFGs and TAGs can only generate links between terminal symbols in the same elementary structure.

Becker et al. (1991) developed these two assumptions into the notion of *derivational generative capacity* as an alternative to weak generative capacity and tree generative capacity.

Definition 1. A *linked string* is a pair $\langle w, \sim \rangle$, where w is a string and $\sim$ is a symmetric binary relation between string positions of w.

Definition 2. We say that a grammar G generates a linked string $\langle w, \sim \rangle$ if G generates w such that $i \sim j$ if and only if the ith and jth symbol of w are generated in the same derivation step.

Definition 3. The *derivational generative capacity* (DGC) of G is the set of all linked strings generated by G.

We notate linked strings either using arcs as above, or sometimes boxed numbers (①, ②, ...) if the linking relation is transitive. In this terminology, then, we would say that CFG lacks the DGC to capture cross-serial dependencies in Dutch. Becker et al. push this approach further: they extend the notion of DGC to a large class of formalisms called *linear context-free rewriting systems* (LCFRSs) and then prove that "doubly unbounded" German scrambling is beyond the power of this entire class.

The merit of this approach is that it eschews notions of generative capacity which are inappropriate (WGC, tree generative capacity) or vague (SGC) in favor of a notion (DGC) which allows a rigorous result to be proven from minimal assumptions (namely, what the correct dependencies are and how dependencies must be represented in CFG and LCFRS). It might be objected that it adds an *ad hoc* notion of generative capacity instead of working with established notions, but it is reasonable to test a grammar formalism according to a notion of generative capacity that is suited to the problem against which it is being tested. Indeed, there may be as many notions of generative capacity as there are applications of formal grammars.

1.3 Basic framework

Our framework generalizes that of Joshi and Becker et al. in two ways: first, just as they introduced a notion of generative capacity suited to a particular question, we take the position, following Miller (1999), that SGC should always be tested with reference to a particular *interpretation domain*. Second, they defined DGC on a class of grammar formalisms with well-defined domains of *locality*, and we extend to a still larger class of formalisms, simple literal movement grammars (Groenink, 1997).

1.3.1 A relativized view of SGC

Miller (1999) deals with the elusiveness of SGC by defining it not as the set of structural descriptions of a grammar, but the set of *interpretations* of its structural descriptions in a particular *interpretation domain*, for example, constituency or dependency (see Figure 1.6). Thus SGC must always be considered with respect to an interpretation domain. This creates many notions of SGC, but ensures that the SGC of two formalisms can be meaningfully compared—provided they both have interpretation functions in a common domain.

A strength of Miller's approach is that it tries to preserve the generality of the formalisms that can be dealt with, placing no restrictions on structural descriptions or their interpretation functions except that interpretation functions are not supposed to add information to structural descriptions

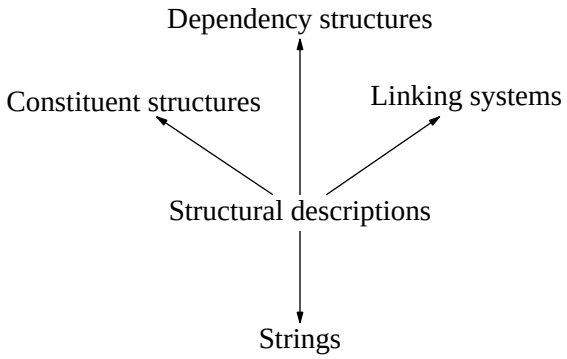


Figure 1.6: Structural descriptions, strings, and interpretations under Miller's framework.

that is not specified by the formalism. A formalism might even have multiple possible interpretation functions for a single domain. For example, we might or might not want a constituency interpretation to consider terminal nodes to be constituents. There would then be multiple possibilities for the SGC of the formalism with respect to a single domain (Miller, 1999, pp. 13–14). But this generality makes it hard to make definite statements about what a formalism is capable of. Characterizing the SGC of a formalism with respect to a domain using one particular interpretation function will not tell us whether a different interpretation function might have made the formalism more or less powerful.

The approach of Joshi and of Becker et al. allows a more uniform treatment. Miller cites their work with approval, rightly observing that it can be subsumed into his framework as one among many interpretation domains. But the key to the TAG approach, which sets it apart from Miller's, is the concept of *locality*. Their definition of DGC assumes that structural descriptions can be decomposed into elementary structures, and specifies a *local* interpretation of each: all the symbols in each elementary structure are linked, and no others. This completely determines the interpretation functions for a large class of formalisms. Thus they can claim, using a consistent set of assumptions, that Dutch cross-serial dependencies are possible for TAG but impossible for CFG, or that German scrambling is impossible for any LCFRS.

We generalize the TAG approach to allow for many interpretation domains, but our interpretation domains will continue to specify the *local interpretation functions* on elementary structures. When an elementary structure has more than one possible local interpretation function, we leave the choice up to the grammar (a departure from Miller) and assume that the formalism contains all the possibilities. For example, we could relax the definition of DGC so that not all symbols in an elementary structure are necessarily linked. Then the choice of which symbols are linked would be specified in the grammar, as in our example above of a TAG with links.

In the following section we define local interpretation functions more precisely.

1.3.2 Simple literal movement grammars

There are many grammar formalisms for which interpretation functions can be decomposed into local interpretation functions on elementary structures; all the formalisms we will consider are subclasses of Groenink's *simple literal movement grammar*, or sLMG (Groenink, 1997).

Definition 4. A *simple LMG* is a tuple $\langle T, N, V, S, A, P\rangle$, where:

- T is a finite set of *terminal symbols* and N is a finite set of *nonterminal symbols*
- V is a set of *variables*
- $S \in N$ is called the *start symbol*
- A is a set of axioms of the form

  $$X(\alpha_1, \ldots, \alpha_m)$$

  where $X \in N$ and $\alpha_j \in T^*$
- P is a set of productions of the form

  $$X(\alpha_1, \ldots, \alpha_m) :- Y_1(\beta_{11}, \ldots, \beta_{1m_1}), \ldots, Y_n(\beta_{n1}, \ldots, \beta_{nm_n})$$

  ("the α_i are an X if the β_{1i} are a Y_1 and the β_{2i} are a Y_2, etc.") where

- $X, Y_i \in N$
- $\alpha_j, \beta_{ij} \in (T \cup V)^*$
- each β_{ij} consists of a single variable, and
- each variable in the production appears at least once on the right-hand side and at least once on the left-hand side

This is equivalent to range concatenation grammar or RCGs (Boullier, 2000), the difference being that RCG variables stand for ranges of positions of a fixed input string, whereas LMG variables simply stand for strings. Thus LMGs can be thought of as a special type of Post system (Post, 1943; Chomsky and Miller, 1963).

Definition 5. A *linear sLMG* is an sLMG in which for each production, each variable in the production appears exactly once on the right-hand side and exactly once on the left-hand side.

Linear sLMG is equivalent to LCFRS, nonerasing multiple context-free grammar (Seki et al., 1991), local scattered context grammar (Rambow and Satta, 1999), and simple RCG (Boullier, 2000).

Definition 6. Let π be an sLMG production, $x_1, \ldots, x_n$ be the distinct variables occurring in π, and $w_1, \ldots, w_n \in T^*$. Then $\pi[w_1/x_1, \ldots, w_n/x_n]$ *instantiates* or is an *instantiation* of π.

Definition 7. If G is an sLMG, we say that G *derives* $X(\alpha_1, \ldots, \alpha_m)$ according to the following recursive definition:

- An axiom $X(\alpha_1, \ldots, \alpha_m)$ of G is derivable by G.
- $X(\alpha_1, \ldots, \alpha_m)$ is derivable by G if there is a production in G which can be instantiated as

$$X(\alpha_1, \ldots, \alpha_m) \coloneq Y_1(\beta_{11}, \ldots, \beta_{1m_1}), \ldots, Y_n(\beta_{n1}, \ldots, \beta_{nm_n})$$

  and $Y_i(\beta_{i1}, \ldots, \beta_{im_i})$ is derivable by G for $1 \leq i \leq n$.

- Nothing else is derivable by G.

The weak generative capacity $L(G)$ of G is defined to be $\{w \mid G \text{ derives } S(w)\}$.

Definition 8. A *derivation* of an sLMG G is a tree or term over (names of) productions of G. The set of derivations of G is defined recursively:

- If $\pi = X(\alpha_1, \ldots, \alpha_m)$ is an axiom of G, $\pi()$ is a derivation of $X(\alpha_1, \ldots, \alpha_m)$.
- If a production π of G can be instantiated as

$$X(\alpha_1, \ldots, \alpha_m) :\!- Y_1(\beta_{11}, \ldots, \beta_{1m_1}), \ldots, Y_n(\beta_{n1}, \ldots, \beta_{nm_n})$$

  and τ_i is a derivation of $Y_i(\beta_{i1}, \ldots, \beta_{im_i})$, $1 \leq i \leq n$, then $\pi(\tau_1, \ldots, \tau_n)$ is a derivation of $X(\alpha_1, \ldots, \alpha_m)$.

To add interpretations, we effectively change an sLMG from a definition of a predicate into a definition of a multivalued function, similar to attribute grammars (Knuth, 1968).

Definition 9. A *functional sLMG* is an sLMG $\langle T, N, V, S, A, P\rangle$ together with:

- for each $X \in N$ and arity n, a domain D_X^n (not necessarily disjoint)
- for each axiom $X(\alpha_1, \ldots, \alpha_m) \in A$, a constant *local interpretation* $d \in D_X^m$
- for each production in P of the form

$$X(\alpha_1, \ldots, \alpha_m) :\!- Y_1(\beta_{11}, \ldots, \beta_{1m_1}), \ldots, Y_n(\beta_{n1}, \ldots, \beta_{nm_n}),$$

  a *local interpretation function* $f : D_{Y_1}^{m_1} \times \cdots \times D_{Y_n}^{m_n} \to D_X^m$

Definition 10. We say that G *derives* $X(\alpha_1, \ldots, \alpha_m)$ *with* an *interpretation* according to the following recursive definition:

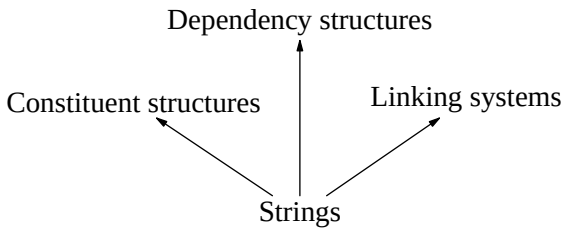


Figure 1.7: Strings and interpretations under functional sLMG.

- An instantiation of an axiom of G with local interpretation d is derivable by G with interpretation d.

- $X(\alpha_1, \ldots, \alpha_n)$ is derivable by G with interpretation d if there is a production in G with function f which can be instantiated as

$$X(\alpha_1, \ldots, \alpha_m) :- Y_1(\beta_{11}, \ldots, \beta_{1m_1}), \ldots, Y_n(\beta_{n1}, \ldots, \beta_{nm_n})$$

  and $Y_i(\beta_{i1}, \ldots, \beta_{im_i})$ is derivable by G with interpretation d_i for $1 \leq i \leq n$, and where $d = f(d_1, \ldots, d_n)$.

- Nothing else is derivable by G.

Then the strong generative capacity $\Sigma(G)$ of G (with respect to the domain D_S^1) is defined to be

$$\{d \mid G \text{ derives some string } w \text{ with interpretation } d\}.$$

(Note that whereas Miller's interpretation functions map from structural descriptions to interpretations (Figure 1.6), these functional sLMGs map directly from strings to interpretations (Figure 1.7), the structural descriptions not appearing explicitly.)

We will make use occasionally of a trivial interpretation function, the string yield function:

Definition 11. The *string yield function* is the interpretation function defined in terms of the following local interpretation functions: to each production

$$X(\alpha_1, \ldots, \alpha_m) :- Y_1(\beta_{11}, \ldots, \beta_{1m_1}), \ldots, Y_n(\beta_{n1}, \ldots, \beta_{nm_n})$$

associate the local interpretation function defined on tuples of strings:

$$f(\langle\beta_{11}, \ldots, \beta_{1m_1}\rangle, \ldots, \langle\beta_{n1}, \ldots, \beta_{nm_n}\rangle) = \langle\alpha_1, \ldots, \alpha_m\rangle$$

If the sLMG is nonlinear, this pattern may not always match; in such cases f is undefined.

Many formalisms also have a *tree yield function*, but its definition depends on the definition of the formalism. For example, a CFG's tree yield function would generate CFG derivation trees; a TAG's tree yield function would generate derived trees as in Figure 1.2.

* * *

Our definition of interpretation functions gives enough control to prove results with genuine relevance to applications. Because they are defined with reference to particular domains, we can test the right properties of a formalism; because they are defined in terms of local interpretation functions, we can firmly characterize a formalism's SGC.

In the main chapters of this dissertation, we will use this framework to define interpretation domains suited to particular applications and then compare various grammar formalisms in those interpretation domains. The goal is to see whether more results like those of Joshi and Becker et al. can be obtained in these other areas of application, and what implications they have for those applications.

Chapter 2

Foundational issues

This chapter contains various building blocks which will be used in later chapters. Section 2.1 contains definitions of all the grammar formalisms used in the thesis; it may be safely skipped and referred to as necessary. Section 2.2 deals with issues related to parsing which the discussion of Section 3.1 depends on.

2.1 Grammars

2.1.1 Tree-adjoining grammars

We have already introduced tree-adjoining grammars in Section 1.2, but what follows is a more precise definition which also includes the substitution operation.

Definition 12. An *auxiliary tree* is a finite tree with a distinguished frontier node called its *foot node*, which we mark with the symbol $*$. The path from an auxiliary tree's root node to its foot node is called its *spine*.

Definition 13. A *tree-adjoining grammar* (Joshi, Levy, and Takahashi, 1975; Joshi and Schabes, 1997) is a tuple $\langle T, N, I, A, S \rangle$, where

- T is a finite set of terminal symbols
- N is a finite set of nonterminal symbols, $N \cap T = \emptyset$

- $N' = N \times \{\epsilon, \text{NA}, \text{OA}\}$ is the set of nonterminal symbols with *adjoining constraints*; unless otherwise indicated, equivalence is understood to be modulo adjoining constraints
- I is a finite set of *initial trees*, which are finite trees whose interior labels are drawn from N' and whose frontier labels are drawn from $N' \cup T$
- A is a finite set of auxiliary trees whose interior labels are drawn from N', whose frontier labels are drawn from $N' \cup T$, and whose root and foot nodes bear the same label
- $S \subseteq I$ is a set of initial trees which can begin a derivation

Definition 14. The result of *adjoining* an auxiliary tree β with root/foot label X at a node η with label X is the tree obtained as follows: detach the subtree rooted by η and call it γ_η, leaving behind a copy of η; attach β by merging its root node with (the copy of) η; reattach γ_η by merging its root node with the foot node of β.

Definition 15. The result of *substituting* an initial tree α with root label X at a frontier node η with label X is the tree obtained by merging the root node of α with η.

Definition 16. A *derived tree* (or *derived initial tree* or *derived auxiliary tree*) of G is obtained by taking an elementary tree γ in S (or I or A, respectively), and:

- substituting a derived initial tree at each of the non-foot frontier nonterminal nodes (called *substitution nodes*)
- adjoining a derived auxiliary tree at each of the nodes with adjoining constraint OA and zero or more of the nodes without adjoining constraint NA

Definition 17. The tree set or tree generative capacity of a TAG G is the set of all possible derived trees of G. The string set or weak generative capacity of G is the set of yields of derived trees of G.

The following three restrictions of TAG have been proposed to capture some of the additional descriptive power of TAG while remaining weakly context-free.

No adjunction: tree substitution grammars

Definition 18. A *tree-substitution grammar* or TSG (Schabes, 1990) is a tree-adjoining grammar with no auxiliary trees.

As a historical note, TAG as originally defined only had adjunction; substitution was introduced later by Abeillé (1988), and then adjunction was dropped by Schabes et al. (1988) to form TSG, though it does not seem to have been called by that name until later (Schabes, 1990).

No wrapping adjunction: tree-insertion grammars

Definition 19. A *left* (or *right*) *auxiliary tree* is an auxiliary tree in which every frontier node to the right (or left, respectively) of the foot node is labeled with the empty string.

Definition 20. A *tree-insertion grammar* or TIG (Schabes and Waters, 1993; Schabes and Waters, 1995), originally termed a "lexicalized context-free grammar," is a TAG in which all auxiliary trees are either left or right auxiliary trees, and adjunction is constrained so that:

- no left (right) auxiliary tree can be adjoined on any node that is on the spine of a right (left) auxiliary tree, and
- no adjunction is permitted on a node that is to the right (left) of the spine of a left (right) auxiliary tree.

Limited spine adjunction: regular form

In his original definition, the details of which we omit here, Rogers (1994) defines a restriction on TAG adjunction, called *regular adjunction*, that can generate only regular path sets. He then identifies a subclass of TAGs, called TAGs in *regular form*, which have the property that every derived tree that can be derived using unrestricted adjunction could also have been derived using only regular adjunction. But since Rogers' recognition algorithm only performs regular adjunction,

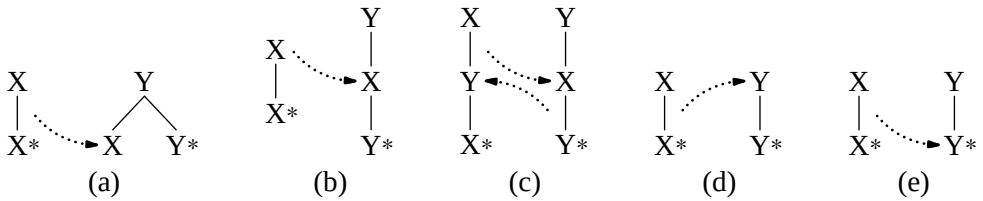


Figure 2.1: Examples of adjunction in regular-form TAG. (a) Off-spine adjunction, allowed; (b) acyclic spine adjunction, allowed; (c) cyclic spine adjunction, not allowed; (d) root adjunction, not allowed; (e) foot adjunction, allowed.

it cannot in general produce all possible derivations of a sentence and therefore cannot be used as a parser.

A more technical issue is that regular adjunction can occur at either the root or foot, which creates derivational ambiguity. Rogers' algorithm, however, cannot distinguish between the two. If we want the parser to compute derivations, one or the other should be disallowed. Following Schuler et al. (2000), we prohibit adjunction at the root.[1] This leads us to the following definition, which narrows Rogers' definition to eliminate both of the above problems:

Definition 21. We say that a TAG is in *regular form*, or an RF-TAG, if there exists some partial ordering $\preceq$ over nonterminal symbols such that if β is an auxiliary tree whose root and foot nodes are labeled X, and η is a node labeled Y on β's spine where adjunction is allowed, then $X \preceq Y$, and $X = Y$ only if η is a foot node.

Thus adjunction at nodes not lying along the spine and adjunction at the foot node are allowed freely; adjunction at nodes lying along the spine is allowed to a bounded depth, but adjunction at the root is not allowed at all (see Figure 2.1).

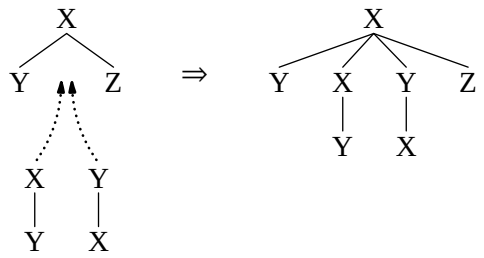


Figure 2.2: Sister adjunction.

Multiple adjunction and sister adjunction

In Schabes and Shieber's extended notion of derivation (1994), a distinction is made between *modifier auxiliary trees* and *predicative auxiliary trees*. Multiple modifier auxiliary trees may be adjoined at a single node, but only one predicative auxiliary tree may be adjoined at a single node. We combine this idea with an operation borrowed from d-tree substitution grammar (Rambow, Vijay-Shanker, and Weir, 1995) called *sister-adjunction*:

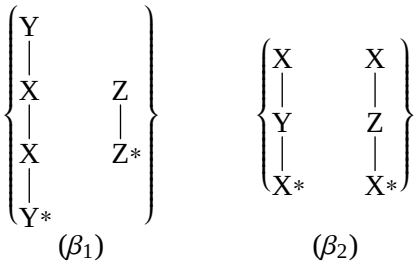
Definition 22. The result of *sister-adjoining* an initial tree α under a node η at position i is the tree obtained by

- if $i = 0$: adding α as the leftmost daughter of η;
- if $0 < i < n$, where n is the number of daughters of η: inserting α between the ith and $(i+1)$st daughter of η;
- if $i = n$: adding α as the rightmost daughter of η.

See Figure 2.2. As in Schabes and Shieber's extension, multiple trees may be sister-adjoined at the same position (Chiang, 2000).

This extension does not add any weak generative power. However, a TAG extended in this way is no longer an sLMG, strictly speaking, because its derivation trees can have unbounded branching

[1]If we had prohibited adjunction at the foot, as is more customary, and allowed adjunction at the root, then the resulting grammars would not be coverable by CFGs (see Section 2.2.2). It might be possible to relax the definition of a cover grammar to allow this, but we do not pursue this possibility here.



(β_1) (β_2)

Figure 2.3: Example multicomponent TAG elementary tree sets.

factors, whereas an sLMG's derivation trees only allow bounded branching.

2.1.2 Multicomponent grammars

Multicomponent TAGs (Weir, 1988) are TAGs whose elementary structures are sets of elementary trees. The basic operation is the simultaneous adjunction or substitution of all the trees in a set. In *set-local* multicomponent TAG, all the trees must compose into the same elementary tree set; in *tree-local* multicomponent TAG, all the trees must compose into the same elementary tree. For example, Figure 2.3 shows some multicomponent TAG elementary tree sets. In a set-local multicomponent TAG, β_1 would be able to adjoin into β_2, by adjoining the first component into the first component and the second component into the second component. Moreover, β_2 would be able to adjoin into β_1, by adjoining both components into the two X nodes of the first component. But in a tree-local multicomponent TAG, the former would not be possible because the two adjunction sites are in different components.

We may generalize this concept to sLMGs in general.

Definition 23. A *multicomponent predicate* is one whose arguments are partitioned into one or more *components* (shown separated by a colon):

$$X(\alpha_{11}, \ldots, \alpha_{1m_1} : \ldots : \alpha_{n1}, \ldots, \alpha_{nm_n})$$

The *dissolution* of a multicomponent predicate with the above form is the set

$$\{X_1(\alpha_{11}, \ldots, \alpha_{1m_1}), \ldots, X_n(\alpha_{n1}, \ldots, \alpha_{nm_n})\}$$

Definition 24. A *set-local* (or *component-local*) *multicomponent production* is an sLMG production whose predicates are multicomponent predicates, and all the arguments of each right-hand-side component (or predicate, respectively) are found in a single component of the left-hand side.

The *dissolution* of a multicomponent production is the set of all possible well-formed sLMG productions formed out of the dissolutions of its predicates (keeping left-hand-side predicates on the left-hand side and right-hand-side predicates on the right-hand side).

Definition 25. A *set-local* (or *component-local*) *multicomponent sLMG* is an sLMG with set-local (or component-local, respectively) multicomponent productions and a single-component start predicate.

If a formalism $\mathcal{F}$ can be embedded in sLMG, then *set-local* (or *component-local*) *multicomponent* $\mathcal{F}$ consists of set-local (or component-local, respectively) multicomponent sLMGs such that the dissolution of each production is a well-formed production of $\mathcal{F}$.

A component-local multicomponent TAG is more commonly known as a *tree-local multicomponent TAG*. Set-local multicomponent CFG is also known as *local scattered-context grammar* (Rambow and Satta, 1999).

Proposition 1. *If a formalism $\mathcal{F}$ can be embedded in sLMG, then component-local multicomponent $\mathcal{F}$ is weakly equivalent to $\mathcal{F}$.*

Proof. Observe that in the dissolution of a component-local multicomponent production, all the components of each right-hand-side predicate end up in the same production. Therefore, given a component-local multicomponent sLMG G, we can dissolve G and augment the nonterminal alphabet to guarantee that it has the same behavior as the original grammar. Let P' be the set of all productions that can be obtained as follows: for any $\pi \in P$, relabel the left-hand predicate to be π

itself and relabel each right-hand predicate to a production with a matching left-hand side. Then let P'' be the union of the dissolutions of all the productions in P'. This set of productions forms a grammar of $\mathcal{F}$ weakly equivalent to G. □

2.2 Parsing

2.2.1 Computing derivation forests

The basic parsing algorithm for sLMGs is straightforward: since they are just deductive systems, a chart-based deductive parser (Shieber, Schabes, and Pereira, 1995) can operate on an sLMG fairly transparently. Such a parser essentially searches the space of all possible instantiations of the productions of the input grammar for an instantiation with left-hand side $S(w)$, where w is the input string. Because the variables of an sLMG, as in an RCG, can only be instantiated to substrings of w, the number of instantiations of each sLMG production, and therefore the running time of the parser, is polynomial in $|w|$.

To obtain a parse forest, that is, a representation of all possible parses of a given string, we can use a construction due to Bertsch and Nederhof (2001), a generalization of similar constructions for other formalisms (Bar-Hillel, Perles, and Shamir, 1964; Vijay-Shanker and Weir, 1993). Given an sLMG G and an input string w, it computes another sLMG which compactly represents all possible derivations of w by G. (If G is linear, then the construction works for a general finite-state automaton, as shown by Bertsch and Nederhof.)

Define a new nonterminal alphabet

$$N' = \bigcup_{X \in N, m \leq f} \{X\} \times Q^{2m}$$

where $Q = \{0, \ldots, |w|\}$ and f is the maximum arity of any predicate. Intuitively, each nonterminal $X \in N$ and arity m is accompanied by $2m$ string positions, indicating the left and right input positions of the arguments of X. Next define a new set of productions P': for each production

$\pi \in P$, if π has the form

$$X_0(\alpha_{01}, \ldots, \alpha_{0m_0}) \coloneq X_1(\alpha_{11}, \ldots, \alpha_{1m_1}), \ldots, X_n(\alpha_{n1}, \ldots, \alpha_{nm_n})$$

for each α_{ij}, define a series of indices $q(i, j, 0), \ldots, q(i, j, |\alpha_{ij}|) \in Q$, which are the input positions between the symbols of α_{ij}. Then let

$$X'_i = \langle X_i, q(i, 1, 0), q(i, 1, |\alpha_{i1}|), \ldots, q(i, m_i, 0), q(i, m_i, |\alpha_{im_i}|)\rangle$$

for all possible values of $q(i, j, k) \in Q$, subject to the following constraints:

- if the kth symbol of α_{ij} and the k'th symbol of $\alpha_{i'j'}$ are the same variable, then $q(i, j, k-1) = q(i', j', k'-1)$ and $q(i, j, k) = q(i', j', k')$;
- if the kth symbol of α_{ij} is a terminal symbol a, then $q(i, j, k-1)+1 = q(i, j, k)$ and $w_{q(i,j,k)} = a$.

These constraints ensure that the $q(i, j, k)$ correspond to a valid instantiation of π and are consistent with the input string. From these X'_i construct the production

$$X'_0(\alpha_{01}, \ldots, \alpha_{0m_0}) \coloneq X'_1(\alpha_{11}, \ldots, \alpha_{1m_1}), \ldots, X'_n(\alpha_{n1}, \ldots, \alpha_{nm_n})$$

These productions form a new grammar G_w whose start symbol is $\langle S, 0, |w|\rangle$. Observe that even if G was nonlinear, G_w is essentially a CFG; the nonterminals do all the work and the arguments do not further constrain the derivations. The size of G_w is $O(|G|n^{(r+1)f})$ productions, where $|G|$ is the number of productions, n is the length of w, f is the maximum arity, and r is the maximum number of nonterminals on the right-hand side of a production. Therefore the running time of this construction is also $O(|G|n^{(r+1)f})$.

This algorithm is by no means optimal, however. In order to improve parsing time, one would have to write a specialized parser for the grammar or, equivalently, construct a *cover grammar* for it. For example, in order to achieve $O(n^3)$ time complexity for parsing CFGs, we must either

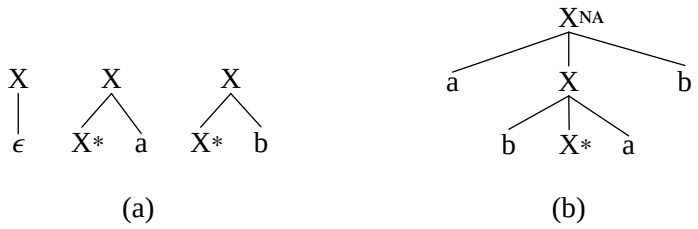


Figure 2.4: Example of weakly context-free TAG.

convert the CFG into Chomsky normal form (that is, strict binary-branching) or use a specialized algorithm like Earley's algorithm, which effectively binarizes the grammar on the fly. In either case, there must be a way of reconstructing the derivations of the original grammar. We discuss this topic further below.

2.2.2 Cover grammars

When considering the use of new formalisms, we are especially interested increasing a formalism's power in some desirable respect while preserving it in some costly respect. The most familiar example of a cost criterion is WGC: Joshi (2003) speaks of "squeezing" SGC out of a formalism without increasing its WGC. From a theoretical standpoint, such formalisms are interesting because they point to finer-grained ways of measuring formal power than the traditional measure of WGC. A more practical cost criterion would be parsing complexity: it would be ideal to gain extra SGC without increasing the asymptotic complexity of the parsing algorithm.

These two constraints, WGC and parsing complexity, often coincide: proofs of weak equivalence to, say, CFG, are often accompanied by $O(n^3)$ parsing algorithms. Indeed, we argue that to have one without the other is not likely to be very interesting.[2] For example, the TAG of Figure 2.4a generates a CFL ($\{a, b\}^*$), and adding the tree in Figure 2.4b does not increase its WGC, nor would

[2] A notable exception would the Lambek calculus (Lambek, 1958), which is weakly context-free (Pentus, 1993) but NP-complete to parse (Pentus, 2003), because the Lambek-to-CFG conversion does not preserve derivations.

adding any tree (or multicomponent tree set) over {a, b}. Conversely, the language

$$\{a^i b^j c^k \mid ijk + 1 \text{ is a prime number}\}$$

has an $O(n^3)$ recognition algorithm but whatever formal system generates it is not likely to resemble a CFG.

It would be easier to characterize what it means to "squeeze" SGC out of a formalism if we used a tighter constraint, one which entailed both preservation of WGC and preservation of parsing complexity. Such a constraint is suggested by examining the parsing algorithms for common weakly context-free formalisms. For example, the parsers for RF-TAG and TIG are based on CKY; their items are of the form $[X, i, j]$ and are combined in various ways, but always according to the deductive rule schema

$$\frac{[X, i, j] \quad [Y, j, k]}{[Z, i, k]}$$

where the material below the line is deduced from the material above the line (Shieber, Schabes, and Pereira, 1995). But this is just like the CKY parser for CFG in Chomsky normal form. In effect the parser dynamically converts the RF-TAG or TIG into an equivalent Chomsky-normal-form CFG—each parser rule of the above form corresponds to the rule schema $Z \rightarrow XY$.

More importantly, given a grammar G and a string w, a parser can reconstruct a packed forest of all possible derivations of w in G by storing some information inside its chart items. Every time it generates a new item, it takes the derivation information in the antecedent items to compute some new information for the new item. If we think of the parser as dynamically converting G into a CFG G', then we may think of these computations as attached not to the deductive rules of the parser, but to the productions of G'. Indeed, we may think of them as a kind of interpretation function for G' into the domain of G-derivations. We call G' a *cover grammar* for G, following Nijholt (1980). This covering relationship is a relationship between grammars and not parsing strategies: while this covering relationship prescribes a particular approach to parsing G, it is independent of any particular parsing strategy for G'.

So far we have not articulated how the derivations of G are reconstructed. A parser like CKY builds a packed parse forest by storing in each parser item $[X, i, j]$ a set of productions $X \to YZ$ together with *back-pointers* to forests of derivations of Y and Z. Crucially, because these subforests would be exponential in size if unpacked, the parser never accesses their internals; it only deals with back-pointers to them, without dereferencing them. Parsers which use cover grammars also typically use back-pointers in this way, which we formalize as follows:[3]

Definition 26. A *cover sLMG* G' is a functional sLMG whose local interpretation functions f operate on tuples of derivations and are each definable as:

$$f(\langle t_{11}, \ldots, t_{1m_1}\rangle, \ldots, \langle t_{n1}, \ldots, t_{nm_n}\rangle) = \langle u_1, \ldots, u_m\rangle$$

where the t_{ij} are variables and each u_i is drawn from the set τ, which is a set of derivation fragments recursively defined as follows:

1. t_{ij} is in τ (copying a back-pointer)
2. $\pi(\tau_1, \ldots, \tau_n)$ is in τ, where $\tau_i \in \tau$ and π is an sLMG production (creating a derivation fragment with back-pointers)

Definition 27. We say that a cover sLMG G' *covers* another sLMG G if there is a one-to-one correspondence between derivations of G' and derivations of G such that G' derives w with interpretation δ if and only if δ is the corresponding G-derivation, and δ is a derivation of w. We say that the cover *respects* an interpretation domain D if corresponding derivations also have the same interpretation in D.

Definition 28. We say that a formalism $\mathcal{F}'$ *covers* another formalism $\mathcal{F}$ (respecting D) if for any grammar G provided by $\mathcal{F}$, there is a grammar provided by $\mathcal{F}'$ which can cover G (respecting D).

Proposition 2. *TSG, TIG, RF-TAG, and component-local multicomponent CFGs are all coverable by CFG. Component-local multicomponent TAG is coverable by TAG.*

[3] This notion is similar to generalized syntax-directed translation (Aho and Ullman, 1971).

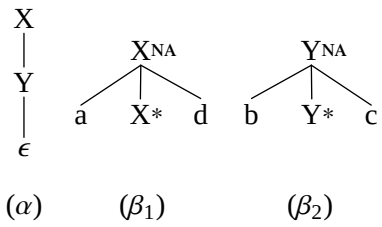


Figure 2.5: Example TAG to be covered. Here adjunction at foot nodes is allowed.

Production	Name
$S(x_1y_2y_2x_2) :\!-\ X(x_1, x_2), Y(y_1, y_2)$	(α)
$X(\mathrm{a}x_1, x_2\mathrm{d}) :\!-\ X(x_1, x_2)$	(β_1)
$X(\epsilon, \epsilon)$	(ϵ_X)
$Y(\mathrm{b}y_1, y_2\mathrm{c}) :\!-\ Y(y_1, y_2)$	(β_2)
$Y(\epsilon, \epsilon)$	(ϵ_Y)

Figure 2.6: sLMG representation of the TAG of Figure 2.5.

Proof. Implicit in the parsing algorithms for these formalisms. □

As an example, a CFG which covers the RF-TAG of Figure 2.5 is shown in Figure 2.7. The nonterminals of this grammar consist of an elementary tree name, a superscripted tree address, a dot indicating the "top half" or "bottom half" of the node (to prevent multiple adjunctions at a node), and a stack in square brackets. When a tree β is adjoined into another tree γ, γ is pushed onto the stack so that it can be recalled when β is finished; however, if β is adjoined at the foot node of γ, then γ does not need to be recalled, so it is not pushed onto the stack, as in the programming-language technique of tail recursion.

The notion of a cover grammar provides a new view of the question posed by Joshi (2003), "How much strong generative power can be squeezed out of a formal system without increasing its weak generative power?" In our present framework, we must understand SGC to be relative to some interpretation domain. Moreover, in light of the foregoing arguments, we should modify the

Production	Local interpretation function	Comment
$S \to \alpha^{0\bullet}$	$\alpha(t_1, t_2) \leftarrow\!\shortmid \langle t_1, t_2\rangle$	
$\alpha^{0\bullet} \to \alpha^{0}_{\bullet}$	$\langle \epsilon_X(), t_2\rangle \leftarrow\!\shortmid \langle -, t_2\rangle$	no adjunction
$\alpha^{0}_{\bullet} \to \alpha^{1\bullet}$	$\langle -, t_2\rangle \leftarrow\!\shortmid \langle -, t_2\rangle$	
$\alpha^{1\bullet} \to \alpha^{1}_{\bullet}$	$\langle -, \epsilon_Y()\rangle \leftarrow\!\shortmid \langle -, -\rangle$	no adjunction
$\alpha^{1}_{\bullet} \to \epsilon$	$\langle -, -\rangle$	
$\alpha^{0\bullet} \to \beta^{0}_{1}[\alpha^{0}]$	$\langle \beta_1(t_1), t_2\rangle \leftarrow\!\shortmid \langle t_1, t_2\rangle$	adjoin β_1
$\beta^{0}_{1}[\alpha^{0}] \to \mathrm{a}\,\beta^{2}_{1}[\alpha^{0}]\,\mathrm{d}$	$\langle t_1, t_2\rangle \leftarrow\!\shortmid \langle t_1, t_2\rangle$	
$\beta^{2}_{1}[\alpha^{0}] \to \beta^{0}_{1}[\alpha^{0}]$	$\langle \beta_1(t_1), t_2\rangle \leftarrow\!\shortmid \langle t_1, t_2\rangle$	adjoin β_1 by "tail recursion"
$\beta^{2}_{1}[\alpha^{0}] \to \alpha^{0}_{\bullet}$	$\langle \epsilon_X(), t_2\rangle \leftarrow\!\shortmid \langle -, t_2\rangle$	no adjunction, return to α
$\alpha^{1\bullet} \to \beta^{0}_{2}[\alpha^{1}]$	$\langle -, \beta_2(t_2)\rangle \leftarrow\!\shortmid \langle -, t_2\rangle$	adjoin β_2
$\beta^{0}_{2}[\alpha^{1}] \to \mathrm{b}\,\beta^{2}_{2}[\alpha^{1}]\,\mathrm{c}$	$\langle -, t_2\rangle \leftarrow\!\shortmid \langle -, t_2\rangle$	
$\beta^{2}_{2}[\alpha^{1}] \to \beta^{1}_{2}[\alpha^{1}]$	$\langle -, \beta_2(t_2)\rangle \leftarrow\!\shortmid \langle -, t_2\rangle$	adjoin β_2 by "tail recursion"
$\beta^{2}_{2}[\alpha^{1}] \to \alpha^{1}_{\bullet}$	$\langle -, \epsilon_Y()\rangle \leftarrow\!\shortmid \langle -, -\rangle$	no adjunction, return to α

Figure 2.7: CFG cover of the TAG of Figure 2.5. Here we leave the local interpretation functions anonymous; $y \leftarrow\!\shortmid x$ denotes the function which maps x to y.

constraint on WGC to be a constraint on coverability. This provides a more rigid framework in which Joshi's question can be explored.

We will show in later chapters that in some interpretation domains, formalisms that are coverable by CFG can indeed have greater SGC than CFG. For example, in Section 5.3.1 we show that RF-TAG can generate a linked string set that CFG cannot:

$$L = \left\{ \begin{array}{c} \\ \text{c a a} \cdots \text{a c b} \cdots \text{b b} \\ \\ \end{array} \right\}$$

In a sense, this greater SGC is "squeezed" out of CFG for free. But this kind of "squeezing" has its limits.

Proposition 3. *If G' covers G, then for any interpretation of G, there exists an equivalent interpretation of G'.*

Proof. The equivalent interpretation for G' is easy to construct: the basic idea is that wherever a local interpretation function f' in the cover generates a G-production π, we substitute π's local interpretation function in place of π. □

This means that we could in fact construct an interpretation for the cover CFG of Figure 2.7 that generates L. It is only under the restriction that links be defined within local domains that L is impossible for CFG.

In Proposition 3, the interpretation of G could even be another cover, which implies that coverability is transitive. This means that while a covered formalism might have greater SGC than the cover formalism in some domains, it can never have greater SGC in the domain of covered derivations.

Corollary 4. *A formalism $\mathcal{F}'$ can cover another formalism $\mathcal{F}$ if and only if $\mathcal{F}'$ can cover every sLMG that $\mathcal{F}$ can.*

In other words, one cannot "squeeze" a formalism a second time to get still more power out. Therefore the class of sLMGs coverable by CFG represents the maximum amount of SGC that can be "squeezed" out of CFG as we have defined it.

In an earlier paper (Chiang, 2001) we tried to characterize this class of grammars more directly by choosing an interpretation domain and exhibiting a formalism that maximized SGC with respect to this domain. But since there are many different ways to do this, it is more fruitful to consider Joshi's question with reference to a particular application, as in the following chapters.

Chapter 3

Statistical parsing

We now turn to *statistical parsing*, in which some probability model defined over parse structures (standardly, phrase-structure trees) is used to determine the best structure or best k structures of a given string. We introduce *weighted* interpretation domains and show how parsing models of a very general nature can be expressed as weighted grammars. But we find that this domain classifies formalisms rather coarsely: any two formalisms with the same coverability (respecting parse structures) also define the same parsing models.

Though this result makes the hope rather dim of squeezing statistical-modeling power out of PCFG for free, it invites a reinterpretation of lexicalized PCFG models (Charniak, 1997; Collins, 1997) as cover grammars of grammars with richer structural descriptions than phrase-structure trees. As a demonstration of this new view, we define a probabilistic TAG model and discuss techniques for obtaining adequate models from corpora which, from this point of view, are labeled only with partial structural descriptions. This model performs at the same level as lexicalized PCFG parsers and captures the same kinds of dependencies they do in a conceptually simpler way.

3.1 Measuring statistical-modeling power

At first glance one might think that flexibility in specifying probability distributions over parses would be the desideratum for statistical parsing, but in fact nearly the opposite is true, because a probability model is learned from a limited amount of data, and we need to *constrain* the set of

possible probability distributions so that a good one can be induced from the data. What actually matters, then, is flexibility in specifying *parameterized* parsing models.

3.1.1 Weighted interpretation domains

Below we examine two of the most commonly-used model types to arrive at a simple way of measuring statistical-modeling power. For generality's sake, we will define conditional models $P(Y \mid X)$, where Y ranges over $\mathcal{Y}$ and X ranges over $\mathcal{X}$. In a conditional parsing model, Y would be the parse tree and X would be the input string, and the parsing problem would be to compute $\arg\max_Y P(Y \mid X)$. Alternatively, one could use a *generative* parsing model, in which $Y = \langle S, T \rangle$ is the parse tree T including the input string S, and X is some fixed value. Then the parsing problem would be to compute

$$\begin{aligned} \arg\max_T P(T \mid S) &= \arg\max_T \frac{P(S,T)}{P(S)} \\ &= \arg\max_T P(S,T) \end{aligned}$$

In either case the training data are represented by a distribution $\tilde{p}(x, y)$.

History-based models A conditional *history-based model* (Black et al., 1992; Collins, 1999) is one which decomposes each outcome y into a sequence of decisions $d_1 \cdots d_n$, where the d_i are drawn from some set $\mathcal{D}$. Each decision is conditioned on x and the preceding decisions; in order to reduce the number of contexts, there are functions Φ_x which group histories $d_1 \cdots d_{i-1}$ into equivalence classes c. Then the parameters of the model are $p(d \mid c)$, and

$$P(y \mid x) = \prod_i p(d_i \mid \Phi_x(d_1 \cdots d_{i-1})) \tag{3.1}$$

Given a training sample $\tilde{p}(x, y)$, it is easy to get the maximum-likelihood parameter estimate subject to the constraint

$$\sum_d p(d \mid c) = 1 \tag{3.2}$$

for all c: it is simply

$$\hat{p}(d \mid c) = \frac{1}{Z_c} \sum_x \sum_{\substack{\mathbf{d}_1, \mathbf{d}_2 \in \mathcal{D}^* \\ \Phi_x(\mathbf{d}_1) = c}} \tilde{p}(x, \mathbf{d}_1 d \mathbf{d}_2) \tag{3.3}$$

where Z_c is a normalization constant.

Maximum-entropy models A conditional *maximum-entropy model* (Berger, Della Pietra, and Della Pietra, 1996) defines a set of arbitrary *feature* functions $f_i : \mathcal{X} \times \mathcal{Y} \to \mathbb{Z}_0^+$. Given a training sample $\tilde{p}(x, y)$, we seek the maximum-entropy probability distribution $p(y \mid x)$ subject to the constraints

$$E_{\tilde{p}}(E_p(f_i \mid x)) = E_{\tilde{p}}(f_i) \qquad \text{for all } i \tag{3.4}$$

and

$$\sum_y p(y \mid x) = 1 \qquad \text{for all } x \tag{3.5}$$

This distribution can be shown to have the following form:

$$p(y \mid x) = \frac{1}{Z_x} \prod_i \theta_i^{f_i(x,y)} \tag{3.6}$$

where the θ_i are real-valued feature weights and the Z_x are normalization factors which are found by solving (3.4) and (3.5), respectively. We can also go in the reverse direction: starting with (3.6),

the maximum-likelihood estimate subject to the constraint (3.5) turns out to be the same as the solution to (3.4).

Equation (3.4) does not in general have an analytic solution for the θ_i. There are various numerical methods for doing this (for example, iterative-scaling (Darroch and Ratcliff, 1972; Berger, Della Pietra, and Della Pietra, 1996); Malouf (2002) provides an overview and comparison of several methods), but they are all more computationally expensive than the relative-frequency estimator for history-based models.

Conclusion Since the both the above cases model the probability of a parse as the product of parameter values, we can characterize the statistical-modeling power of a grammar by its SGC with respect to the domain of parameter vectors, or, more generally still, semiring weights. This definition can be coupled with other interpretation domains in the obvious way to provide interpretations in the domain of weighted strings, weighted trees, and so on.

3.1.2 Grammar-based statistical parsing models

Below we show how to define statistical parsing models based on grammars. First we define grammars with generalized weights (Goodman, 1999):

Definition 29. If R is a semiring, an *R-weighted sLMG* is one whose interpretation functions have composition operations of the form

$$\phi(p_1, \ldots, p_n) = p \prod p_i \tag{3.7}$$

where $p \in R$.

Next we instantiate this definition for both of the above classes of models. A history-based model decomposes parses into sequences of decisions; a maximum-entropy model represents them as vectors of features. When either model is based on an sLMG, we require that its parse representations respect the derivation structure of the grammar.

History-based models For a history-based model based on a *linear* sLMG G, we define a conditional history-based model of *productions*, $P(\pi \mid X, n)$, where π ranges over productions, X over their left-hand nonterminals, n over their arities. This model decomposes each production π into a decision sequence $d_1 \cdots d_n$; we assign this sequence to π as its weight. The weights are multiplied by concatenation,[1] which induces a history-based model of *derivations* in which a derivation's decomposition is simply its weight. The reason G must be linear is that otherwise the derivation distribution would not sum to one in general; but even when G is linear it is not guaranteed (Booth and Thompson, 1973).

In the prototypical case, the production model decomposes each production trivially, that is, into itself. Thus a PCFG is a history-based model based on a CFG whose production model decomposes a production $X \to \alpha$ into itself. Probabilistic TAG (Resnik, 1992; Schabes, 1992) can be constructed in a similar way, although the details depend on how we embed TAG into sLMG.

Maximum-entropy models For a maximum-entropy model based on a grammar G, we do not define a model of productions, but we do define feature functions on productions. The feature functions assign to each production π a feature vector; we assign this vector to π as its weight. The weights are multiplied by vector addition, which induces a maximum-entropy model of derivations in which a derivation's feature vector is simply its weight.

Note that the above definition does not allow totally arbitrary features, but it does allow them to be arbitrary within a local domain. Note also that it applies even to nonlinear sLMGs. In addition to providing a general way of defining maximum-entropy models on grammars, it provides a new way of efficiently training maximum-entropy parsing models. We devote the remainder of this section to this latter point.

Above we noted that estimating a maximum-entropy model requires the computation of the expected feature values, which in turn requires a summation over all possible parses. Since there will in general be exponentially many parses, it is impractical to perform this summation by brute

[1] The fact that concatenation is noncommutative and our semirings are commutative does not matter, because of the independence assumption between productions.

force. For joint models, random sampling of the parse space seems to be the only recourse (Abney, 1997); for conditional models the problem is not as severe (Johnson et al., 1999), but the number of parses is still asymptotically exponential. Non-grammatical solutions include modeling moves of a pushdown automaton (Ratnaparkhi, 1997), which, however, is susceptible to the label-bias problem (Lafferty, McCallum, and Pereira, 2001), and reranking the parses of a history-based model (Collins, 2000).

But it would seem most straightforward to compute expected feature values the same way we typically compute the most probable parse of a sentence: use a dynamic-programming algorithm that exploits the structure of the grammar derivation space. Miyao and Tsujii (2002) describe an algorithm in which parameters are estimated from a *feature forest*, a compact representation of the feature vectors of all the possible parses of S. It is essentially a generalization of conditional random fields (Lafferty, McCallum, and Pereira, 2001) to CFGs (more accurately, and-or graphs). But they do not specify how these feature forests are obtained. Clark and Curran have done so for CCG (Clark and Curran, 2003), and Miyao et al. have done so with HPSG (Miyao, Ninomiya, and Tsujii, 2003); our definition of maximum-entropy models based on sLMGs makes this possible in a much more general way.

Miyao and Tsujii's algorithm is essentially the same as the Inside-Outside algorithm (Baker, 1979; Lari and Young, 1990), which computes the expected number of times $E(\pi)$ that each production π will be used in a CFG. Let F be a derivation forest of an sLMG for a given string. This F can be thought of as a weighted CFG. Then

$$E(f_i \mid S) = \sum_{\pi} f_i(\pi) E(\pi) \tag{3.8}$$

where π ranges over productions in F and the expectation $E(\pi)$ is computed by the Inside-Outside algorithm using the weights of F (it makes no difference algorithmically that F is not a proper PCFG), and then dividing by the normalization constant Z_S from (3.6). This algorithm only works on forests with a finite number of derivations, although it is possible to extend it to the general

case (Goodman, 1999).

Geman and Johnson (2002) have employed a different kind of packed representation to define dynamic-programming algorithms for estimating stochastic unification-based grammars. In a separate paper (Chiang, 2003) we compared their representation with Miyao and Tsujii's and concluded that while the former is much more general, the latter is more efficient for the common case.

3.1.3 The coarse-grainedness of statistical-modeling power

With the above definitions characterizing statistical-modeling power in hand, we may ask whether we can "squeeze" statistical-modeling power out of a formalism for free—that is, without affecting coverability. The answer is the following negative result:

Proposition 5. *If a grammar G' covers G respecting some interpretation domain D, then the R-weighted versions of G' and G are equivalent with respect to the R-weighted version of D.*

Proof. Let G_w be the weighted version of G. Given G', it is easy to construct an R-weighted version of G' that is equivalent to G_w: replace each composition operation f' in G' by (3.7), with p equal to the product of the weights in G_w of the composition operations invoked by f'. The resulting weighted grammar G'_w is equivalent to G_w, for even though G'_w and G_w multiply the production weights together in different orders, the result is the same because R is commutative. □

This means that squeezing SGC while preserving coverability will not increase the number of models that can be described. On the other hand, it is trivially true that a formalism with greater SGC with respect to parse structures can describe more models. For example, TIG (see Section 2.1), because it can be covered by CFG respecting strings, cannot be used to describe any more models of string probability than CFG can. But because it has greater tree generative capacity than CFG, it can describe more models of tree probability.

Therefore, since many recent parsing models (Charniak, 1997; Collins, 1997; Miller et al., 2000) are based on PCFG, we should not expect to improve much on them by moving to TIG, and not at all by moving to TSG or RF-TAG. However, this result reveals a deeper insight into

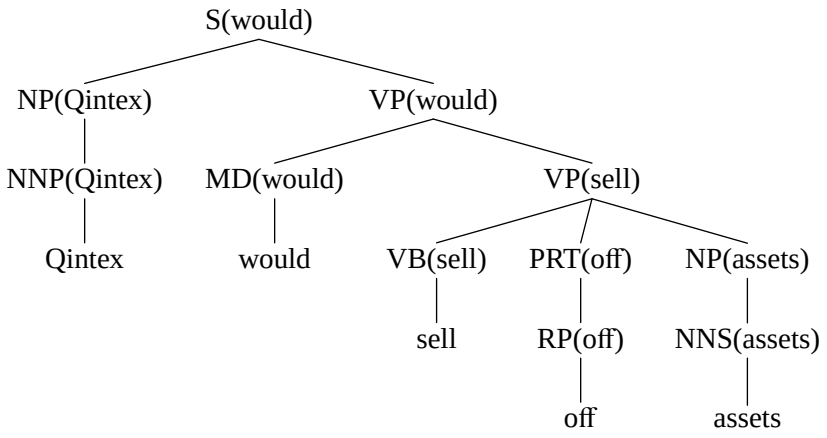


Figure 3.1: Lexicalized PCFG tree.

these models. The CFGs they are based on are called *lexicalized*, which means that they have been modified so that every nonterminal symbol contains a lexical item (see Figure 3.1 for an example derivation). Before training, a set of rules (Magerman, 1995) is used to identify a *head child* for every node in the training data; then the training data are transformed in the following way, bottom-up:

1. If a node labeled X has a head child which is a terminal node labeled w, then relabel the node $X(w)$.

2. If a node labeled X has a head child labeled $Y(w)$, relabel the node $X(w)$.

The conventional wisdom regarding lexicalized PCFG is that it rearranges the lexical information in trees so that it can be used more effectively; specifically, it places pairs of words into the same local domain so that *bilexical statistics* can be collected. But experiments have shown (Gildea, 2001; Klein and Manning, 2003; Bikel, 2004a) that bilexical statistics actually help very little in parsing. We argue below that the lexicalization process does more than rearrange lexical information to create bilexical dependencies.

Recall that a cover grammar has pieces of the covered derivations attached to its productions, and it uses information transmitted through its nonterminal symbols to ensure that the pieces are

attached correctly. For example, one way to construct a cover grammar for a TSG (see Section 2.1) could be:

1. For every non-substitution node labeled X with address η in an elementary tree α, add as a decoration the singleton set
$$\{\langle \alpha, \eta \rangle\}$$

2. For every substitution node labeled X, add the decoration
$$\{\langle \alpha, \epsilon \rangle \mid \alpha \text{ is an elementary tree with root label } X\}$$

3. For every node labeled X with decoration $\{\langle \alpha, \eta \rangle\}$ immediately dominating nodes labeled $X_1, \ldots, X_n$ with decoration $\Delta_1, \ldots, \Delta_n$, construct the CFG rules
$$X(\alpha, \eta) \rightarrow X_1(\alpha_1, \eta_1), \ldots, X_n(\alpha_n, \eta_n) \qquad \text{for all } \langle \alpha_i, \eta_i \rangle \in \Delta_i$$

If we further assume that each elementary tree α has a single lexical anchor w_α, then observe that (α, η) subsumes (w_α), so that the CFG with rules
$$X(w_\alpha) \rightarrow X_1(w_{\alpha_1}), \ldots, X_n(w_{\alpha_n}) \qquad \text{for all } \langle \alpha_i, \eta_i \rangle \in \Delta_i$$

generates an approximate superset of the original TSG. But this grammar is none other than a lexicalized PCFG. We may therefore think of a lexicalized PCFG as an approximate cover of a TSG. If each elementary tree has a unique lexical anchor and each node in a tree has a unique label, then the cover is exact.

Collins' models decompose the generation of productions more finely; we omit the details here, only noting that the use of a Markov process to generate adjuncts makes an infinite number of productions possible. To avoid infinite grammars, we add sister-adjunction (see Section 2.1), which can create new children under a node in a manner similar to Collins' Markov model.

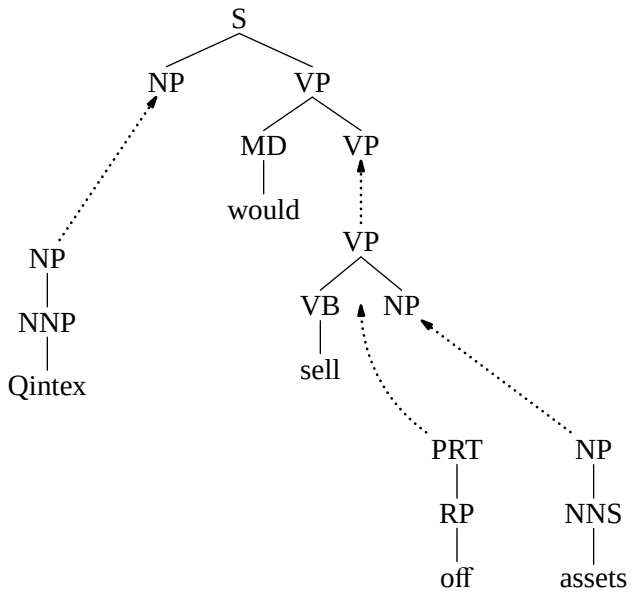


Figure 3.2: Derivation of TSG with sister-adjunction corresponding to tree of Figure 3.1.

Therefore Collins' Model 2, minus its distance measure, can alternatively be thought of as defined on a TSG with sister-adjunction (see Figure 3.2). Since a TSG's derivations are distinct from its derived trees and contain more information than them, we should likewise think of Collins' Model 2 as being defined over richer structural descriptions than are found in the Penn Treebank, and we should think of the lexicalization process as reconstructing information rather than rearranging information, and this information is structural rather than lexical.

In the following section we build a parsing model from the ground up according to this new perspective. This perspective also suggests an alternative to reconstruction heuristics: to treat training as a partially-unsupervised learning problem and use EM to train the model from partial structural descriptions.

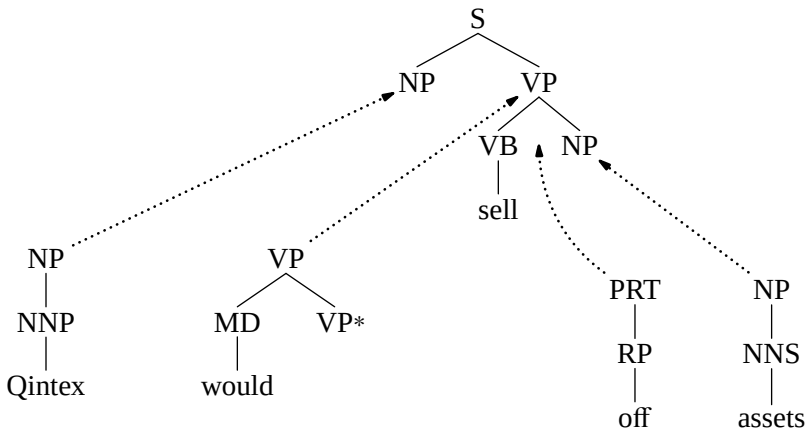


Figure 3.3: Derivation of TIG with sister-adjunction corresponding to tree of Figure 3.1.

3.2 A probabilistic TIG model

If lexicalized CFG is a cover grammar for something like a TAG, then perhaps a statistical parsing model should be defined directly as one. In this section we define a model based on probabilistic TIG with sister-adjunction. Such a formulation would have several advantages (beyond any aesthetic considerations).

First, we noted that lexicalized CFG is only an approximate cover because it uses lexical anchors as a proxy for their elementary structures. Various modifications to lexicalized CFG have been found to improve parsing accuracy, for example:

- an S which does not dominate an argument NP to the left of its head is relabeled SG, so that the attachment of the clause can be conditioned on whether it has a subject (Collins, 1999)
- an NP which does not dominate another NP which does not dominate a POS is relabeled NPB, and if its parent is not an NP, an NP node is inserted (Collins, 1999) because PCFG mismodels Treebank-style two-level NPs (Johnson, 1998)
- every node's label is augmented with the label of its parent (Charniak, 2000; Johnson, 1998)

Such changes are not always obvious *a priori* and often must be devised anew for each language

or each corpus. But the above modifications are not necessary in a TAG-like model, because it has statistics of pairs of elementary trees and not just pairs of words. Thus many dependencies that have to be stipulated in a PCFG by tree transformations are captured for free in a probabilistic TAG model.

Second, TAG provides greater flexibility in defining heuristics. For example, we might want elementary trees that contain both a preposition and the head word of the prepositional object, in the hope that the latter will help make PP attachment decisions (Collins and Brooks, 1995). Or, in a sentence with an auxiliary verb, like the above example "Qintex would sell off assets," we might want the subject to attach to the tree for 'sell' instead of the tree for 'would.' It would be tricky to make such changes to a head word percolation scheme, but easy with a TAG or TIG (see Figure 3.3 for a TIG derivation of the latter example).

Third, by decoupling the reconstruction heuristics from the training process proper, this new view suggests alternative training methods. Below we describe experiments using Expectation-Maximization to train directly on the observed, not the reconstructed data (a method explored previously by Hwa (1998) for TIG).

3.2.1 Basic definition

The parameters of a probabilistic TAG model (Resnik, 1992; Schabes, 1992) are:

$$\sum_{\alpha} P_i(\alpha) = 1$$

$$\sum_{\alpha} P_s(\alpha \mid \eta) = 1$$

$$\sum_{\beta} P_a(\beta \mid \eta) + P_a(\text{NONE} \mid \eta) = 1$$

where α ranges over initial trees, β over auxiliary trees, and η over nodes. $P_i(\alpha)$ is the probability of beginning a derivation with α; $P_s(\alpha \mid \eta)$ is the probability of substituting α at η; $P_a(\beta \mid \eta)$ is the probability of adjoining β at η; finally, $P_a(\text{NONE} \mid \eta)$ is the probability of nothing adjoining

at η. The probability of a derivation can then be expressed as a product of the probabilities of the individual operations of the derivation.

We restrict the TAG to be a TIG for efficiency reasons. The above model works for TIG just as it does for TAG. However, the original definition of probabilistic TIG (Schabes and Waters, 1996) is flawed because it allows one left auxiliary tree and one right auxiliary tree (but not more than one of each) to adjoin at the same node in either order, but the probability model does not distinguish the two orders, so that the total probability of all valid derivations is greater than one. Hwa (2001, p. 30) describes how to fix the problem, but our fix is simply to prohibit this type of simultaneous adjunction.

Our variant (henceforth TIG-SA) adds another set of parameters for sister-adjunction:

$$\sum_{\alpha} P_{sa}(\alpha \mid \eta, i, \alpha') + P_{sa}(\text{STOP} \mid \eta, i, \alpha') \;=\; 1$$

where α and α' range over initial trees, and (η, i) over possible sister-adjunction sites. Let n be the number of children of η; $P_{sa}(\alpha \mid \eta, i, \alpha')$, $0 \leq i \leq n$, is the probability of sister-adjoining α under η at position i, when α' is the previous[2] tree to have sister-adjoined at that position (or START if none). Thus modifier trees are generated as a first-order Markov process, as in the model used in BBN's SIFT system (Miller et al., 2000) and the base-NP model of Collins (1999).

3.2.2 Independence assumptions and smoothing

Since the number of parameters in this model is too high to get reasonable estimates from corpus data, we generate an elementary tree in two steps and smooth each step: first the *tree template* (that

[2]Here "previous" means "next closest to the lexical anchor," which presupposes a single lexical anchor; we could alternatively define it to mean "next to the left," which would be more general but less efficient.

is, the elementary tree stripped of its anchor), then the anchor. Thus:

$$P_i(\alpha) = P_{i_1}(\tau_\alpha)P_2(w_\alpha \mid \tau_\alpha)$$

$$P_s(\alpha \mid \eta) = P_{s_1}(\tau_\alpha \mid \eta) \cdot P_2(w_\alpha \mid \tau_\alpha, t_\eta, w_\eta)$$

$$P_a(\beta \mid \eta) = P_{a_1}(\tau_\beta \mid \eta) \cdot P_2(w_\beta \mid \tau_\beta, t_\eta, w_\eta)$$

$$P_{sa}(\alpha \mid \eta, i, \alpha') = P_{sa_1}(\tau_\alpha \mid \eta, i, X_{\alpha'}) \cdot P_2(w_\alpha \mid \tau_\alpha, t_\eta, w_\eta, X_{\alpha'})$$

where τ_α is the tree template of α and w_α is the lexical anchor of α, and similarly for β; w_η is the lexical anchor of the elementary tree containing η, and t_η is the part-of-speech tag of that anchor. We have reduced α' to its root label $X_{\alpha'}$. Note that the same probability P_2 is used for all three composition operations: for adjunction and substitution, $X_{\alpha'}$ is assigned the value START.

These probabilities each have three backoff levels:

	$P_{s_1,a_1}(\gamma \mid \cdots)$	$P_{sa_1}(\alpha \mid \cdots)$	$P_2(w \mid \cdots)$
1	$\tau_\eta, w_\eta, \eta_\eta$	$\tau_\eta, w_\eta, \eta_\eta, i, X_{\alpha'}$	$\tau_\gamma, t_\eta, w_\eta, X_{\alpha'}$
2	τ_η, η_η	$\tau_\eta, \eta_\eta, i, X_{\alpha'}$	$\tau_\gamma, t_\eta, X_{\alpha'}$
3	$\overline{\tau}_\eta, \eta_\eta$	$\overline{\tau}_\eta, \eta_\eta, i$	τ_γ
4	X_η	$\emptyset$	t_γ

where τ_η is the tree template of the elementary tree containing η, $\overline{\tau}_\eta$ is τ_η stripped of its anchor's POS tag, X_η is the label of η, and η_η is the address of η in its elementary tree; τ_γ is the tree template of γ, and t_γ is the POS tag of its anchor. The backed-off models are combined by linear interpolation:

$$e = \lambda_1 e_1 + (1 - \lambda_1)(\lambda_2 e_2 + (1 - \lambda_2)(\lambda_3 e_3 + (1 - \lambda_3)e_4)) \tag{3.9}$$

where e_i is the estimate at level i, and the λ_i are computed by a combination of formulas used by

Collins (1999) and Bikel et al. (1997):

$$\lambda_i = \left(1 - \frac{d_{i-1}}{d_i}\right)\left(\frac{1}{1 + 5u_i/d_i}\right) \tag{3.10}$$

where d_i is the number of occurrences in training of the context at level i ($d_0 = 0$), and u_i is the number of unique outcomes for that context seen in training.

To handle unknown words, we treat all words seen five or fewer times in training as a single symbol UNKNOWN, following Collins (1997).

3.2.3 Parsing

We used a CKY-style TIG parser similar to the one described by Schabes and Waters (1996), with a modification to ensure completeness (because foot nodes are effectively empty, which standard CKY does not handle). We also extended the parser to allow sister-adjunction.

The parser uses a beam search, assigning a score to each item $[\eta, i, j]$ and pruning away any item with score less than 10^{-5} times that of the best item for that span, following Collins (1997). The score of an item is its inside probability multiplied by the prior probability $P(\eta)$, following Goodman (1997). $P(\eta)$, in turn, is decomposed as $P(\overline{\tau}_\eta \mid t_\eta, w_\eta) \cdot P(t_\eta, w_\eta)$, so that the first term can be smoothed by linear interpolation (as above) with the backed-off estimate $P(\overline{\tau}_\eta \mid t_\eta)$, again following Collins (1999).

As mentioned above, words occurring five or fewer times in training were replaced with the symbol UNKNOWN. When any such word w occurs in the test data, it is also replaced with UNKNOWN. Following Collins (1997), the parser only allows w to anchor templates that have POS tags observed in training with w itself, or templates that have the POS tag assigned to w by MXPOST (Ratnaparkhi, 1996); all other templates are thrown out for w.

When parsing English, we use Collins' comma rule: when the parser combines two constituents, if the right-hand constituent has a comma to its left, it must also have a comma (or the end of the sentence) to its right, or else the two constituents cannot be combined. In our parser,

because a binary-branching cover grammar is used, this means that if a right modifier (substitution or sister-adjunction) has a comma to its left, it must have a comma (or the end of the sentence) to its right; if a left modifier has a comma to its left, then the *parent node* must have a comma to its right.

3.3 Training from partial structural descriptions

We want to obtain a maximum-likelihood estimate of these parameters, but cannot estimate them directly from the Treebank, because the sample space of probabilistic TIG-SA is the space of TIG-SA derivations, not the derived trees that are found in the Treebank. For there are many TIG-SA derivations which can yield the same derived tree, even with respect to a single grammar. We need, then, to reconstruct TIG-SA derivations somehow from the training data.

One approach, taken by Magerman (1995) and others for lexicalized PCFGs and Neumann (1998) and others (Xia, 1999; Chen and Vijay-Shanker, 2000) for TAGs, is to use heuristics to reconstruct the derivations, and directly estimate the probabilistic TIG-SA parameters from the reconstructed derivations. Another approach, taken by Hwa (1998), is to choose some grammar general enough to parse the whole corpus and obtain a maximum-likelihood estimate by Expectation-Maximization. Below we discuss the first approach, and then a combination of the two approaches.

3.3.1 Rule-based reconstruction

Given a CFG and a Magerman-style head-percolation scheme, an equivalent TIG-SA can be constructed whose derivations mirror the dependency analysis implicit in the head-percolation scheme.

For each node η, the head-percolation and argument/adjunct rules classify exactly one child of η as a head and the rest as either arguments or adjuncts. We use Magerman and Collins' rules with few modifications (see Tables 3.4 and 3.5), but we treat coordination specially: if an X dominates a CC, and the rightmost CC has an X to its left and to its right, then that CC is marked as the head and the two nearest Xs on either side as arguments, and no further rules apply.

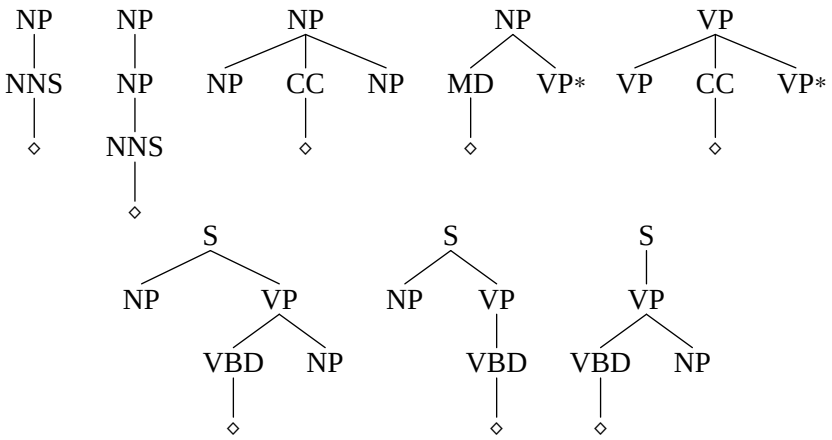


Figure 3.4: Examples of frequently-occurring extracted tree templates. ◇ marks where the lexical anchor is inserted.

Using this classification into heads, arguments, and adjuncts, we can construct a TIG-SA derivation (including elementary trees) from a derived tree as follows:

1. If η is an adjunct, excise the subtree rooted at η to form a sister-adjoined initial tree.

2. If η is an argument, excise the subtree rooted at η to form an initial tree, leaving behind a substitution node.

3. But if η is an argument and η' is the nearest ancestor with the same label, and η is the rightmost descendant of η', and all the intervening nodes, including η', are heads, excise the part of the tree from η' down to η to form an auxiliary tree, leaving behind a head node.

Rules (1) and (2) produce the desired result; rule (3) changes things somewhat by making trees with recursive arguments into auxiliary trees. Its main effect is to extract VP auxiliary trees for modal and auxiliary verbs. In the present experiments, in fact, it is restricted to nodes labeled VP. The complicated restrictions on η' are simply to ensure that a well-formed TIG-SA derivation is produced.

When we run this algorithm on sections 02–21 of the Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993), the resulting grammar has 50,628 lexicalized trees (with words seen five

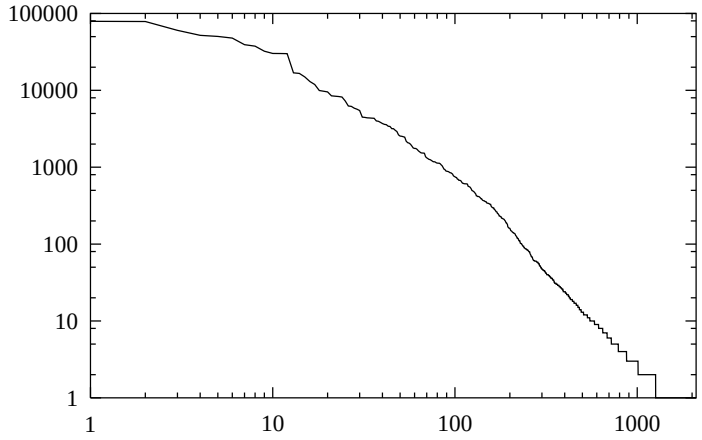


Figure 3.5: Distribution of tree templates: frequency versus rank (log-log)

or fewer times replaced with UNKNOWN). However, if we consider elementary tree templates, the grammar is quite manageable: 2104 tree templates, of which 1261 occur more than once (see Figure 3.5). A few of the most frequent tree templates are shown in Figure 3.4.

So the extracted grammar is fairly compact, but how complete is it? Ideally the size of the grammar would converge, but if we plot its growth during training (Figure 3.6), we see that even after training on 1 million words, new elementary tree templates continue to appear at the rate of about four every 1000 words, or one every ten sentences.

We do not consider this effect to be seriously detrimental to parsing. Since 90% of unseen sentences can be parsed perfectly with the extracted grammar, its coverage is good enough potentially to parse new data with state-of-the-art accuracy. Note that even for the remaining 10% it is still quite possible for the grammar to derive a perfect parse, since there can be many TIG-SA derivations which yield the same derived tree. Nevertheless, we would like to know the source of this effect and minimize it. Three possible explanations are:

- New constructions continue to appear.
- Old constructions continue to be (erroneously) annotated in new ways.

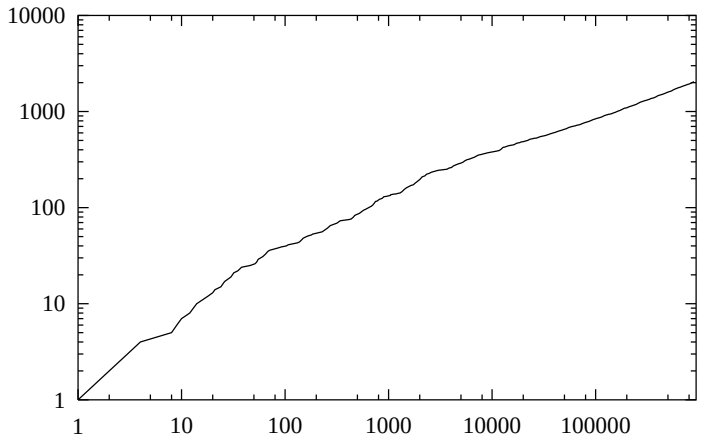


Figure 3.6: Growth of grammar during training: types versus tokens (log-log)

- Old constructions continue to be combined in new ways, and the extraction heuristics fail to factor this variation out.

In a random sample of 100 once-seen elementary tree templates, we found (by casual inspection) that 34 resulted from annotation errors, 50 from deficiencies in the heuristics, and four apparently from errors in the text itself. Only twelve appeared to be genuine.[3]

Therefore the extracted grammar is more complete than Figure 3.6 suggests at first glance. Evidently, however, our extraction heuristics have room to improve. The majority of trees resulting from deficiencies in the heuristics involved complicated coordination structures, which is not surprising, since coordination has always been problematic for TAG. In practice, we throw out all elementary tree templates seen only once in training, on the assumption that they are most likely the result of noise in the data or the extraction heuristics.

This method is extremely similar to that of Xia (1999) and that of Chen (Chen and Vijay-Shanker, 2000), the main difference being that these other methods tend to add brackets to the Treebank in order to obtain a more sensible grammar, whereas our method tends to reproduce the Treebank bracketing more closely, in order to facilitate comparison with other statistical parsers.

[3]This survey was performed on an earlier version of the extraction heuristics.

			≤ 40 words		
Model	LR	LP	CB	0CB	≤ 2 CB
PCFG (Charniak, 1997)	71.7	75.8	2.03	39.5	68.1
Magerman (1995)	84.6	84.9	1.26	56.6	81.4
Charniak (1997)	87.5	87.4	1.0	62.1	86.1
present model	87.7	87.7	1.02	65.2	84.8
Collins (1999)	88.5	88.7	0.92	66.7	87.1
Charniak (2000)	90.1	90.1	0.74	70.1	89.6
			≤ 100 words		
Model	LR	LP	CB	0CB	≤ 2 CB
PCFG (Charniak, 1997)	70.6	74.8	2.37	37.2	64.5
Magerman (1995)	84.0	84.3	1.46	54.0	78.8
Charniak (1997)	86.7	86.6	1.20	59.5	83.2
present model	87.0	87.0	1.21	62.2	82.2
Collins (1999)	88.1	88.3	1.06	64.0	85.1
Charniak (2000)	89.6	89.5	0.88	67.6	87.7

Table 3.1: Parsing results using heuristics on English. LR = labeled recall, LP = labeled precision; CB = average crossing brackets, 0 CB = no crossing brackets, ≤ 2 CB = two or fewer crossing brackets. All figures except CB are percentages.

We trained the model using the extraction heuristics on sections 02–21 and tested it on section 23. The results (Table 3.1) show that our parser lies roughly midway between the earliest (Magerman, 1995) and latest (Charniak, 2000) of the lexicalized PCFG parsers,[4] and that both do considerably better than a PCFG trained on Treebank trees *qua* CFG derivations (Charniak, 1997). While these results are not state-of-the-art, they demonstrate that a probabilistic TIG-SA parser can perform at the same level of accuracy as a lexicalized PCFG parser—or, under our reinterpretation, that lexicalization makes PCFG parsers perform at the same level of accuracy as a probabilistic TIG-SA parser.

We suspect that our model does not match the best of the lexicalized PCFG models because it is not using the larger elementary structures of TIG-SA very robustly. Fine-tuning the backoff model might bring accuracy closer to the state of the art, but it may be more productive to look to maximum-entropy models to provide greater flexibility in choosing model features with different

[4]Note that these figures are an improvement over an earlier version (Chiang, 2000).

amounts of context.

To see how this system would adapt to a different corpus in a different language, we replaced the head rules and argument/adjunct rules with rules appropriate for the Penn Chinese Treebank (Xia et al., 2000). These were adapted from rules constructed by Xia (1999) and are shown in Tables 3.6 and 3.7. We also retained the coordination rule described above.

We also made the following changes to the experimental setup:

- We lowered the unknown word threshold from five to one because the new training set was smaller.
- The POS tagger for unknown words had to be retrained on the new corpus.
- A beam width of 10^{-4} was used instead of 10^{-5}, for speed reasons.
- The comma pruning rule was not used, because it is based on an empirical observation from the English Treebank which does not hold for the Chinese Treebank.

We then trained the parser on sections 001–270 of the Penn Chinese Treebank (84,873 words) and tested it on sections 271–300 (6776 words). To provide a basis for comparison with performance on English, we performed two further tests. First, we trained the parser on sections 001–270 of the English translation of the Penn Chinese Treebank (118,099 words) and tested it on sections 271–300 (10,913) words). Second, we trained the parser on sections 02–03 of the WSJ corpus (82,592 words) and tested it on the first 400 sentences of section 23 (9473 words) with the same settings as the Chinese parser (but with the comma pruning rule). Note that because of the relatively small datasets used, cross-validation would be desirable for future studies.

The results, shown in Table 3.2,[5] show that this parser is quite usable on a language other than the one it was developed on. Indeed, it was the parser used to bootstrap later releases of the Penn

[5]Because of part-of-speech tagging errors on the part of either the corpus or the parser, two sentences are flagged by the scorer as having the wrong length. The standard policy is to treat the sentence as if it were not in either the gold standard file or the test file, but the more rigorous policy used here is to keep the sentence but treat the test file as if it had not guessed any brackets. For the WSJ corpus, the scores are not affected, but in this case, they are affected slightly. Using the standard policy, labeled recall would be 75.8% for sentences of length $\leq$ 100 and 79.2% for sentences of length $\leq$ 40.

		≤ 100 words				
Model	Corpus	LR	LP	CB	0CB	≤ 2 CB
present	WSJ-small	82.9	82.7	1.60	48.5	74.8
present	Xinhua-English	73.6	77.7	2.75	48.6	66.0
present	**Xinhua**	75.3	76.8	2.72	46.0	67.0
		≤ 40 words				
Model	Corpus	LR	LP	CB	0CB	≤ 2 CB
present	WSJ-small	83.5	83.1	1.42	50.4	77.2
present	Xinhua-English	76.4	82.3	1.39	61.4	78.3
present	**Xinhua**	78.4	80.0	1.79	52.8	74.8

Table 3.2: Parsing results using heuristics on Chinese. Abbreviations are as in Figure 3.1. Xinhua: trained on Penn Chinese Treebank sections 001–270, tested on sections 271–300. Xinhua-English: same, but on English translation. WSJ-small: trained on Penn Treebank, Wall Street Journal sections 02–03, tested on section 23, sentences 1–400.

Chinese Treebank, providing rough parses which human annotators can correct up to twice as fast as annotating from scratch (Chiou, Chiang, and Palmer, 2001).

3.3.2 Training by Expectation-Maximization

In the type of approach we have been discussing so far, we use hand-written rules to reconstruct TIG-SA structural descriptions from the partial structural descriptions in training data, and then train the TIG-SA model by maximizing the likelihood of the reconstructed training data according to the model. However, the estimate we get will maximize the likelihood of the reconstructed training data, but not necessarily that of the observed training data itself. In this section we explore the possibility of training a model directly on partial structural descriptions using the method of Expectation-Maximization (Dempster, Laird, and Rubin, 1977); more specifically, a generalization of the Inside-Outside algorithm (Lari and Young, 1990).

This approach follows a very similar experiment by Hwa (1998). The difference between the two is that whereas Hwa begins with a non-linguistically-motivated grammar that is designed to be general enough to generate any bracketing, we use the grammar and initial model induced by the heuristic method of the previous section. For this reason, Hwa's method is not able to

take advantage of the information in the nonterminal labels in the data, but only the bracketing information.

The expectation step (E-step) of the Inside-Outside algorithm is performed by a parser that computes all possible derivations for each parse tree in the training data. This algorithm is analogous to CKY for TAG (Shieber, Schabes, and Pereira, 1995), except instead of items of the form $[\eta, i, j, k, l]$, where η ranges over elementary tree nodes and i, j, k, and l range over positions in the input string, it uses items of the form $[\eta, \theta_{il}, \theta_{jk}]$, where θ_{il} and θ_{jk} range over addresses in the input tree. By contrast, Hwa's E-step uses a standard TIG parser, but discards chart items with spans that cross a bracket in the input tree. But because the TIG parser achieves its cubic time complexity by pretending that foot nodes have zero spans, Hwa's implementation of the E-step will not work correctly if adjunction is allowed at spine nodes other than the root node. Since this type of adjunction is necessary to show that TIG has greater tree generative capacity than CFG, this implementation is not fully general.

The E-step then uses the derivation forest thus produced to compute inside and outside probabilities and uses these, in turn, to compute the expected number of times each decision occurred. Since a TIG-SA derivation forest has the same form as a CFG derivation forest, this computation is identical to the standard Inside-Outside algorithm; it is not necessary to define a specialized algorithm (Schabes, 1992; Hwa, 1998).

For the maximization step (M-step), we obtain a maximum-likelihood estimate of the parameters of the model using relative-frequency estimation, just as in the original experiment, as if the expected values for the complete data were the training data; the only difference is that the expected values may be fractional.

Smoothing presents a special problem: recall that our model uses several backoff levels combined by linear interpolation. There are several ways one might incorporate this smoothing into the reestimation process, and we chose to depart as little as possible from the original smoothing method: in the E-step, we use the smoothed model, and after the M-step, we use the original formula (3.10) to recompute the smoothing weights based on the new counts computed from the

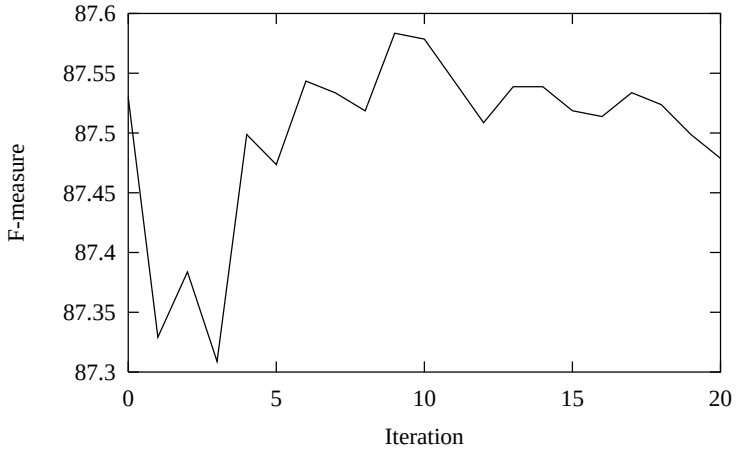


Figure 3.7: Accuracy of reestimated models on held-out data: English, starting with full rule set.

E-step. While simple, this approach has two important consequences. First, since the formula for the smoothing weights intentionally does *not* maximize the likelihood of the training data, each iteration of reestimation is not guaranteed to increase the likelihood of the training data. Second, reestimation tends to increase the size of the model in memory, since smoothing gives nonzero expected counts to many choices which were unseen in training. Therefore, since the resulting model is quite large, if a choice at a particular point in the derivation forest has an expected count below 10^{-15}, we throw it out.

A more theoretically correct method would be to permanently use the smoothing weights computed on the initial model (Bikel, 2004b). This would restore the guarantee of nondecreasing likelihood, and perhaps limit the growth of the model as well. Bikel has performed some initial experiments using this method.

We first trained the initial model on sections 02–21 of the WSJ corpus using the original head rules, and then ran the Inside-Outside algorithm on the same data. We tested each successive model on some held-out data (section 00), using a beam width of 10^{-4}, to determine at which iteration to stop. The F-measure (harmonic mean of labeled precision and recall) for sentences of length ≤ 100 for each iteration is shown in Figure 3.7. We then selected the ninth reestimated model and

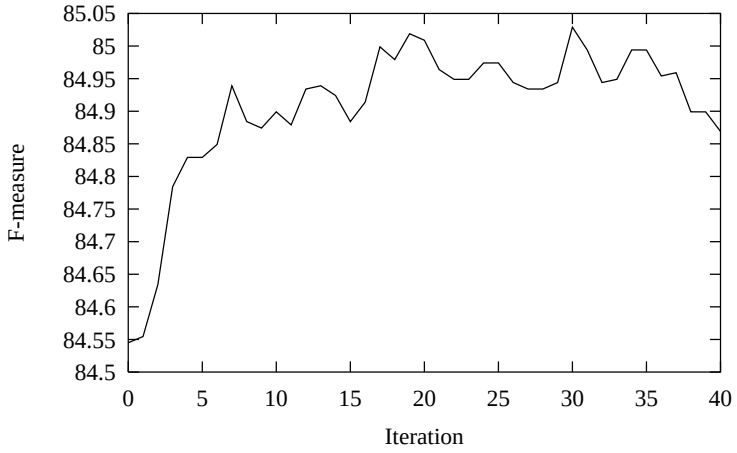


Figure 3.8: Accuracy of reestimated models on held-out data: English, starting with simplified rule set.

compared it with the initial model on section 23 (see Table 3.3). This model did only marginally better than the initial model on section 00, but it actually performs worse than the initial model on section 23. One explanation is that the head rules, since they have been extensively fine-tuned, do not leave much room for improvement. To test this, we ran two more experiments.

The second experiment started with a simplified rule set, which simply chooses either the leftmost or rightmost child of each node as the head, depending on the label of the parent: e.g., for VP, the leftmost child is chosen; for NP, the rightmost child is chosen. The argument rules, however, were not changed. This rule set is supposed to represent the kind of rule set that someone with basic familiarity with English syntax might write down in a few minutes. The reestimated models seemed to improve on this simplified rule set when parsing section 00 (see Figure 3.8); however, when we compared the 30th reestimated model with the initial model on section 23 (see Figure 3.3), there was no improvement.

The third experiment was on the Chinese Treebank, starting with the same head rules used in (Bikel and Chiang, 2000). These rules were originally written by Xia for grammar development, and although we have modified them for parsing, they have not received as much fine-tuning as

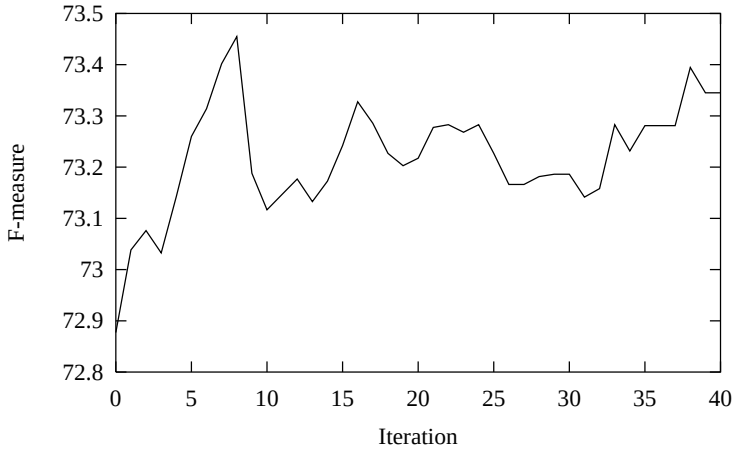


Figure 3.9: Accuracy of reestimated models on held-out data: Chinese, starting with full rule set.

the English rules have. We trained the model on sections 001–270 of the Penn Chinese Treebank, and reestimated it on the same data, testing it at each iteration on sections 301–325 (Figure 3.9). We selected the 38th reestimated model for comparison with the initial model on sections 271–300 (Figure 3.3). Here we did observe a small improvement: an error reduction of 3.4% in the F-measure for sentences of length ≤ 40.

Our hypothesis that reestimation does not improve on the original rule set for English because that rule set is already fine-tuned was partially borne out by the second and third experiments. The model trained with a simplified rule set for English showed improvement on held-out data during reestimation, but showed no improvement in the final evaluation; however, the model trained on Chinese did show a small improvement in both. We are uncertain as to why the gains observed during the second experiment were not reflected in the final evaluation, but based on the graph of Figure 3.8 and the results on Chinese, we believe that reestimation by EM can be used to facilitate adaptation of parsing models to new languages or corpora.

* * *

We have shown how to define parsing models of a general nature based on grammars, and

				≤ 100 words		
Model/Corpus	Step	LR	LP	CB	0CB	≤ 2 CB
English	initial	87.0	87.0	1.21	62.4	82.3
	9	86.4	86.7	1.26	61.4	81.8
English-simple	initial	84.5	84.2	1.54	57.6	78.4
	30	84.2	84.5	1.53	58.0	77.8
Chinese	initial	75.3	76.8	2.72	46.0	67.0
	38	75.2	78.0	2.66	47.7	67.6
				≤ 40 words		
Model/Corpus	Step	LR	LP	CB	0CB	≤ 2 CB
English	initial	87.7	87.8	1.02	65.3	84.9
	9	87.2	87.5	1.06	64.4	84.2
English-simple	initial	85.5	85.2	1.29	60.7	81.1
	30	85.1	85.4	1.30	60.9	80.6
Chinese	initial	78.4	80.0	1.79	52.8	74.8
	38	78.8	81.1	1.69	54.2	75.1

Table 3.3: Parsing results using EM. Original = trained on English with original rule set; Simple = English, simplified rule set. LR = labeled recall, LP = labeled precision; CB = average crossing brackets, 0 CB = no crossing brackets, ≤ 2 CB = two or fewer crossing brackets. All figures except CB are percentages.

found that many of the grammar formalisms that have been proposed to increase the SGC of CFG in fact do not define any more parsing models than PCFG does. We then showed how a lexicalized PCFG can be thought of as a compiled version of a model defined over richer structural descriptions than are found in typical treebanks, and described our implementation of this new view in a probabilistic TIG-SA which performs at the same level of accuracy as lexicalized PCFG. This result demonstrates that TIG-SA is a viable framework for statistical parsing. Moreover, it provides more flexibility than the head- and argument-finding rules of current lexicalized PCFG models. For future work, we would like to explore further how to exploit this flexibility. We would also like to experiment with TAG-based maximum-entropy models as defined above.

Parent	Child
ADJP	first NNS; first QP; first NN; first $; first ADVP; first JJ; first VBN; first VBG; first ADJP; first JJR; first NP; first JJS; first DT; first FW; first RBR; first RBS; first SBAR; first RB; first *
ADVP	last RB; last RBR; last RBS; last FW; last ADVP; last TO; last CD; last JJR; last JJ; last IN; last NP; last JJS; last NN; last *
CONJP	last CC; last RB; last IN; last *
FRAG	last *
INTJ	first *
LST	last LS; last :; last *
NAC	first NN; first NNS; first NNP; first NNPS; first NP; first NAC; first EX; first $; first CD; first QP; first PRP; first VBG; first JJ; first JJS; first JJR; first ADJP; first FW; first *
{NP, NX}	last {NN, NNP, NNPS, NNS, NX, POS, JJR} first NP last {$, ADJP, PRN}; last CD; last {JJ, JJS, RB, QP}; last *
PP	last IN; last TO; last VBG; last VBN; last RP; last FW; last *
PRN	first *
PRT	last RP; last *
QP	first $; first IN; first NNS; first NN; first JJ; first RB; first DT; first CD; first NCD; first QP; first JJR; first JJS; first *
RRC	last VP; last NP; last ADVP; last ADJP; last PP; last *
S	first TO; first IN; first VP; first S; first SBAR; first ADJP; first UCP; first NP; first *
SBAR	first WHNP; first WHPP; first WHADVP; first WHADJP; first IN; first DT; first S; first SQ; first SINV; first SBAR; first FRAG; first *
SBARQ	first SQ; first S; first SINV; first SBARQ; first FRAG; first *
SINV	first VBZ; first VBD; first VBP; first VB; first MD; first VP; first S; first SINV; first ADJP; first NP; first *
SQ	first VBZ; first VBD; first VBP; first VB; first MD; first VP; first SQ; first *
UCP	last *
VP	first TO; first VBD; first VBN; first MD; first VBZ; first VB; first VBG; first VBP; first VP; first ADJP; first NN; first NNS; first NP; first *
WHADJP	first CC; first WRB; first JJ; first ADJP; first *
WHADVP	last CC; last WRB; first *
WHNP	first WDT; first WP; first WP$; first WHADJP; first WHPP; first WHNP; first *
WHPP	last IN; last TO; last FW; last *
X	first *

Table 3.4: Head rules for the Penn (English) Treebank. Rules (delimited by line breaks or semicolons) apply sequentially for each parent node until a match is found. The symbol * stands for any label.

Parent	Child
S	all {NP, SBAR, S} except *A*
{VP, SQ, SINV}	all {NP, SBAR, S, VP} except *A*
SBAR	all S except *A*
SBARQ	all SQ except *A*
NP	all NP except *A*
PP	first {PP, NP, WHNP, ADJP, ADVP, S, SBAR, VP, UCP} after head

where *A* = {-ADV, -VOC, -BNF, -DIR, -EXT, -LOC, -MNR, -TMP, -PRP, -CLR}

Table 3.5: Argument rules for the Penn (English) Treebank. All rules apply to every parent-child pair.

Parent	Child
ADJP	last {ADJP, JJ}; last {AD, NN, CS}; last *
ADVP	last {ADVP, AD}; last *
CLP	last {CLP, M}; last *
CP	last {DEC, SP}; first {ADVP, CS}; last {CP, IP}; last *
DNP	last {DNP, DEG}; last DEC; last *
DVP	last {DVP, DEV}; last *
DP	first {DP, DT}; first *
FRAG	last {VV, NR, NN}; last *
INTJ	last {INTJ, IJ}; last *
LST	first {LST, CD, OD}; first *
IP	last {IP, VP}; last VV; last *
LCP	last {LCP, LC}; last *
NP	last {NP, NN, NT, NR, QP}; last *
PP	first {PP, P}; first *
PRN	last {NP, IP, VP, NT, NR, NN}; last *
QP	last {QP, CLP, CD, OD}; last *
VP	first {VP, VA, VC, VE, VV, BA, LB, VCD, VSB, VRD, VNV, VCP}; first *
VCD	last {VCD, VV, VA, VC, VE}; last *
VRD	last {VRD, VV, VA, VC, VE}; last *
VSB	last {VSB, VV, VA, VC, VE}; last *
VCP	last {VCP, VV, VA, VC, VE}; last *
VNV	last {VNV, VV, VA, VC, VE}; last *
VPT	last {VPT, VV, VA, VC, VE}; last *
UCP	last *
WHNP	last {WHNP, NP, NN, NT, NR, QP}; last *
WHPP	first {WHPP, PP, P}; first *

Table 3.6: Head rules for the Penn Chinese Treebank. Rules (delimited by line breaks or semicolons) apply sequentially for each parent node until a match is found. The symbol * stands for any label.

Parent	Child
VP	all {CP, IP, VP} except -ADV
CP	all {CP, IP} except -ADV
PP	all {NP, DP, QP, LCP, CP, IP, UCP} except -ADV
DNP	all {NP, DP, QP, LCP, PP, ADJP, UCP} except -ADV
DVP	all {NP, DP, QP, VP, ADVP, UCP} except -ADV
LCP	all {NP, DP, QP, LCP, IP, PP, UCP} except -ADV
*	all {-SBJ, -OBJ, -IO, -PRD} except -ADV

Table 3.7: Argument rules for the Penn Chinese Treebank. All rules apply to every parent-child pair. The symbol * stands for any label.

Chapter 4

Applications to language translation

In this chapter we discuss applications of grammars to natural language translation. We define how to measure the power of grammar formalisms for translation and find that this domain classifies formalisms more finely than in the previous chapter: formalisms which were previously equivalent now separate into different levels of power. This means that machine translation stands to gain more from richer grammar formalisms than statistical parsing does; on the other hand, it is all the more important for machine translation research to find the right level of power and not give up on grammar-based methods. We discuss the formal properties of synchronous RF-TAG and the possibility of its use for language translation.

4.1 Measuring translation power

The obvious way to measure the translation power of a grammar formalism is by its SGC with respect to the domain of string pairs: if a grammar formalism can generate more string relations, it will be a more flexible system for defining translations. Tree relations are also a useful measure when constraints on phrase structures are needed.

Grammars which generate pairs of strings or structures are called *synchronous grammars*. The interpretation functions of synchronous grammars are restricted so that transformations occur only at the level of local domains. The most general type of synchronous grammars we will consider

are defined by Bertsch and Nederhof (2001) (called by them "simple range concatenation transducers").

Definition 30. A *synchronous production* is a multicomponent linear sLMG production (see Section 2.1.2) whose predicates have exactly two components, and all the arguments from the *i*th component of each right-hand-side predicate appear in the *i*th component of the left-hand side.

Definition 31. A *synchronous sLMG* is a linear sLMG whose productions are all synchronous productions and with a two-component start predicate S. The string relation defined by a synchronous linear sLMG is the set

$$\{\langle w_1, w_2\rangle \mid S(w_1 : w_2) \text{ is derivable}\}$$

The reason for the linearity requirement is that we would like to be able to take a synchronous grammar generating a relation R and construct an ordinary grammar that generates the projection of R onto one of its components. If the grammar is linear, then it is easy to unify the derivational constraints from both components into a single component, because linear sLMG derivation sets are context-free, and context-free tree sets are closed under intersection.

Any formalism embedded in linear sLMG has a synchronous equivalent. Therefore when we speak of the SGC of a formalism $\mathcal{F}$ with respect to string pairs (or tree pairs), we actually mean the string relation (or tree relation, respectively) generated by the *synchronous* version of $\mathcal{F}$.[1]

4.2 Synchronous grammars for syntax-based translation

Various subclasses of synchronous sLMG have been proposed for machine translation, both non-statistical and statistical. We survey several of them below and compare them according to their SGC with respect to string/tree pairs.

[1] A synchronous version of a grammar is different from a probabilistic version of a grammar in that the former changes the grammar whereas the latter only changes the local interpretations. It would be easy to define synchronous grammars so that they too change only the local interpretations: simply attach the string yield function of one grammar to the productions of another. Such a definition would fit more neatly into our SGC framework, but look less like standard definitions.

4.2.1 Synchronous CFG and TAG

One of the simplest examples of a synchronous grammar is synchronous CFG or syntax-directed translation schemata (Aho and Ullman, 1969). For example, the following grammar fragment would translate between SVO and SOV word orders:

$$\begin{aligned} \mathrm{S}(sp : s'p') &:\!- \mathrm{NP}(s), \mathrm{VP}(p), \mathrm{NP}(s'), \mathrm{VP}(p) \\ \mathrm{VP}(vo : o'v') &:\!- \mathrm{V}(v), \mathrm{NP}(o), \mathrm{V}(v'), \mathrm{NP}(o') \end{aligned} \tag{4.1}$$

Or, in more standard notation, with boxed indices to indicate the correspondence between the two sides:

$$\begin{aligned} &\langle \mathrm{S} \rightarrow \mathrm{NP}\boxed{1}\ \mathrm{VP}\boxed{2}, \mathrm{S} \rightarrow \mathrm{NP}\boxed{1}\ \mathrm{VP}\boxed{2} \rangle \\ &\langle \mathrm{VP} \rightarrow \mathrm{V}\boxed{1}\ \mathrm{NP}\boxed{2}, \mathrm{VP} \rightarrow \mathrm{NP}\boxed{2}\ \mathrm{V}\boxed{1} \rangle \end{aligned} \tag{4.2}$$

In the statistical machine translation literature, synchronous CFGs were proposed by Wu (1997) as *inversion transduction grammars* (Wu, 1997). These are not full synchronous CFGs, but are equivalent to synchronous CFG with a maximum branching factor of two. They have strictly less SGC with respect to string pairs than synchronous CFG does (Aho and Ullman, 1969).

Synchronous TAG was first defined by Shieber (1994) as a correction of an older faulty definition (Shieber and Schabes, 1990). The new definition, which requires that the source and target derivations be isomorphic, is essentially a generalization to TAG of synchronous CFG. In their definition, a synchronous TAG is a set of pairs of elementary trees in which input interior/substitution nodes are coindexed with output interior/substitution nodes.

A synchronous tree-substitution grammar is just a synchronous TAG without adjunction.

Proposition 6. *Synchronous TSG is strongly equivalent to synchronous CFG with respect to string pairs but has greater SGC with respect to tree pairs.*

Proof. Any synchronous TSG can easily be converted into a synchronous CFG generating the

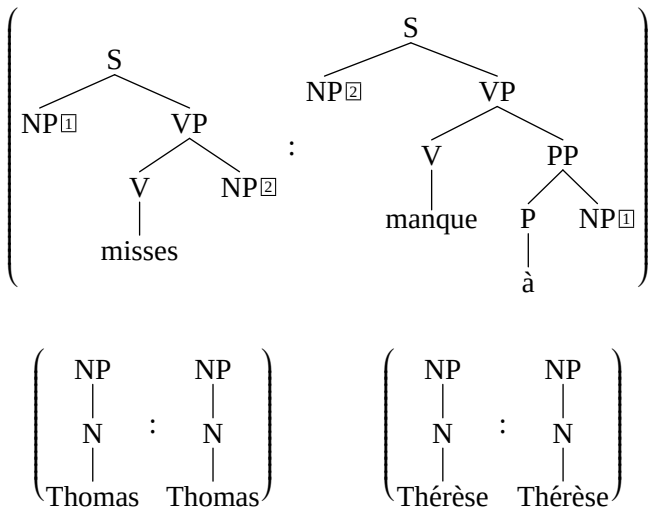


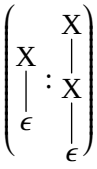
Figure 4.1: Synchronous TSG demonstrating transposition of subject and object.

same string pairs: for each tree pair $\langle \alpha_1 : \alpha_2 \rangle$, create the production pair

$$\langle root(\alpha_1) \rightarrow yield(\alpha_1) : root(\alpha_2) \rightarrow yield(\alpha_2) \rangle$$

where *root* and *yield* are functions returning the root and yield, respectively, of a tree.

On the other hand, the following synchronous TSG generates a trivial example of a tree relation that no synchronous CFG can generate:



because the tree relations generated by a synchronous CFG must always have equal numbers of nonterminal nodes. □

For a linguistic example, see Figure 4.1, which demonstrates how synchronous TSG can be used to transpose subjects and objects, as when translating between the English and French sentences:

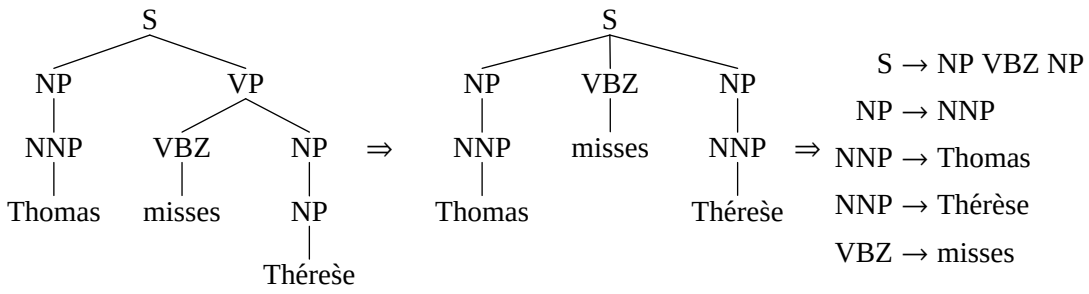


Figure 4.2: Flattening of trees in the translation system of Yamada and Knight (2001).

(4.3) Thomas misses Thérèse.

(4.4) Thérèse manque à Thomas.
Thérèse misses to Thomas

This kind of transformation might also be needed for stylistic reasons, for example, when translating English passive sentences into Chinese, in which the passive is used much less frequently.

Because of its slightly greater power, synchronous TSG has been proposed by Shieber (1994) for generating pairs of TAG derivation trees: to use the terminology of Dras (1999), a synchronous grammar called the *meta-level grammar* (elsewhere in the literature called a *control grammar*) generates pairs of derivation trees of tree-adjoining grammars, called the *object-level* grammars. The resulting system has somewhat more flexibility than synchronous TAG.

More recently, Eisner (2003) and Gildea (2003) have proposed using synchronous TSG by itself for statistical machine translation. However, in a certain sense this had already been achieved by Yamada and Knight (2001). Their system is formally a synchronous CFG in which French productions are generated from English productions (following the naming convention of Brown et al. (1993)) through a sequence of transformations ("noisy channel"). But its trainer flattens English trees by deleting nodes that have the same head word as their parent according to the Magerman rules (see Figure 4.2). Thus their system can transpose subjects and objects as well. It is able to do this within a synchronous CFG because it alters the trees, but we can also reinterpret this model, as we did Collins' parser, as a cover grammar for a synchronous TSG (see Figure 4.3).

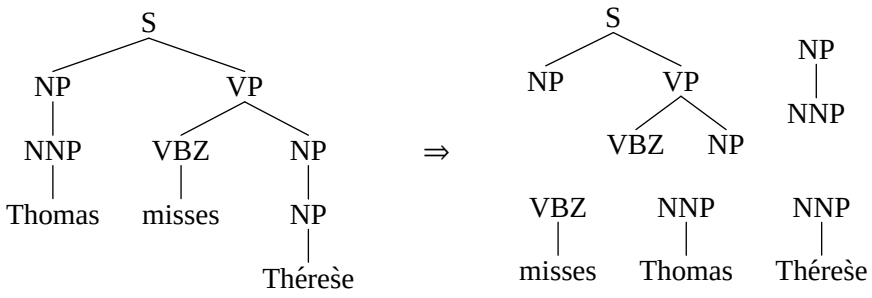


Figure 4.3: Alternative view of flattening operation as extraction of unlexicalized TSG.

4.2.2 Synchronous regular form TAG

Synchronous RF-TAG has been proposed by Dras (1999) as a meta-level grammar for generating pairs of TAG derivation trees, as a further extension to Shieber's suggestion of using TSG. We may also consider synchronous RF-TAG on its own. It has strictly greater SGC with respect to string pairs (and therefore tree pairs) than synchronous CFG does (Chiang, 2002).

Lemma 7 (synchronous pumping lemma). *Let L be a string relation generated by a synchronous CFG. Then there is a constant n such that if $\langle z : z' \rangle$ is in L and $|z| \geq n$ and $|z'| \geq n$, then $\langle z : z' \rangle$ may be written as $\langle uwy : u'w'y' \rangle$, and there exist strings v, x, v', x', such that $|vxv'x'| > 0$, $|vwx| \leq n$, $|v'w'x'| \leq n$, and for all $i \geq 0$, $\langle uv^i wx^i y : u'v'^i w'x'^i y' \rangle$ is in L.*

Proof. The proof is analogous to that of the standard pumping lemma (Hopcroft and Ullman, 1979, pp. 125–127). However, a CFG cannot be put into Chomsky normal form without changing its SGC with respect to string pairs, and there are both sides to take into account. So we let m be the longest right-hand side in either grammar or the number 2, whichever is greater, and k and k' be the size of the two nonterminal alphabets; then $n = m^{kk'}$. This guarantees the existence of a pair of corresponding paths in the derivation of $\langle z : z' \rangle$ such that the same pair of nonterminals $\langle A : A' \rangle$ occurs twice:

$$\langle S : S \rangle \stackrel{*}{\Rightarrow} \langle uAy : u'A'y' \rangle \stackrel{*}{\Rightarrow} \langle u_1 vAxy_1 : u'_1 v'A'x'y'_1 \rangle \stackrel{*}{\Rightarrow} \langle u_1 vwxy_1 : u'_1 v'w'x'y'_1 \rangle$$

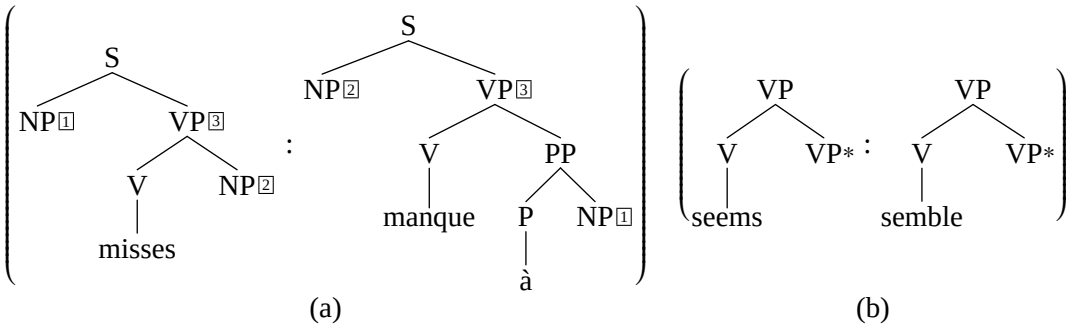


Figure 4.4: Synchronous TAG fragment demonstrating long-distance transposition of subject and object.

If we let $u = u_1 v$ and $y = xy_1$, and likewise $u' = u'_1 v'$ and $y' = x'y'_1$, then $\langle z : z' \rangle = \langle uwy : u'w'y' \rangle$, and for all $i \geq 0$, $\langle uv^i wx^i y : u'v'^i w'x'^i y' \rangle \in L$.

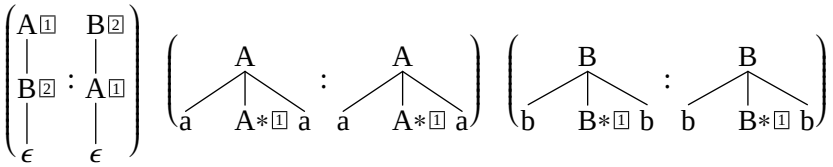
□

Proposition 8. *The string relation*

$$L = \{\langle \mathrm{a}^m \mathrm{b}^n \mathrm{c}^n \mathrm{d}^m : \mathrm{b}^n \mathrm{a}^m \mathrm{d}^m \mathrm{c}^n \rangle\}$$

is generable by a synchronous RF-TAG but not by any synchronous CFG.

Proof. The following synchronous RF-TAG generates L:



But suppose L can be generated by some CFG G. For any n given by the pumping lemma, let $\langle z : z' \rangle = \langle \mathrm{a}^n \mathrm{b}^n \mathrm{c}^n \mathrm{d}^n : \mathrm{b}^n \mathrm{a}^n \mathrm{c}^n \mathrm{d}^n \rangle$ satisfy the conditions of the pumping lemma. Then $vxv'x'$ must contain only a's and d's, or only b's and c's, otherwise $\langle uv^i wx^i y : u'v'^i w'x'^i y' \rangle$ will not be in L. But in the former case, $|vwx| > n$, and in the latter case, $|v'w'x'| > n$, which is a contradiction. □

We saw above how synchronous TSG can be used to transpose subjects and objects. But synchronous TSG cannot perform the same translation with a raising verb added:

(4.5) Thomas seems to miss Thérèse.

(4.6) Thérèse semble manquer à Thomas.
Thérèse seems to miss to Thomas

To do that, we need adjunction of an auxiliary tree pair (Figure 4.4), which stretches the subject-object transposition apart arbitrarily. This is within the power of both TIG and RF-TAG.[2]

Schuler (1999) describes a similar problem translating between Portuguese and English, operating at the level of TAG derivation trees rather than phrase-structure trees. The standard TAG analysis for the sentences

(4.7) John is supposed to have to fly.

(4.8) John is supposed to be going to have to fly.

and so on, would have the auxiliary trees for 'supposed,' 'have,' and 'going' adjoining at VP (see Figure 4.6). Their Portuguese translations, however, have *pressuposto* 'supposed' above the subject, and therefore its auxiliary tree would adjoin at S:

(4.9) É pressuposto que João tem que voar.
it is supposed that John has to fly

(4.10) É pressuposto que João vai ter que voar.
it is supposed that John is going to have to fly

The two derivation tree schemata are shown in Figure 4.7. One solution (Chiang, Schuler, and Dras, 2000) is to use a synchronous RF-TAG to map between the two sets of TAG derivations (Figure 4.8). Again, the displacement of β_1[supposed] gets stretched apart arbitrarily by the interposition of raising verbs.

[2] In our definition of RF-TAG, adjunction is allowed at foot nodes but not auxiliary root nodes, contrary to typical TAG syntactic analyses. We could accept the divergence, or else we could change the definition of RF-TAG to allow adjunction at root nodes instead of foot nodes, but this would make parsing less simple.

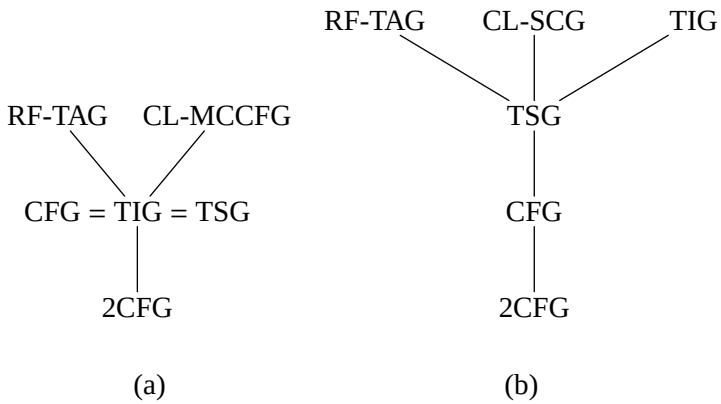


Figure 4.5: Relative translation power of various formalisms: (a) string pairs; (b) tree pairs. An edge connecting two formalisms indicates that the higher has greater SGC than the lower. CL-MCCFG = component-local multicomponent CFG; 2CFG = binary branching CFG.

4.2.3 The fine-grainedness of translation power

Figure 4.5 summarizes the relative SGC of various synchronous grammar formalisms with respect to string and tree pairs, including some results from an earlier paper (Chiang, 2002) not proven here. It is striking that SGC with respect to string pairs and tree pairs classify formalisms quite finely: inversion transduction grammar generates fewer string pairs than synchronous CFG; synchronous CFG generates fewer tree pairs than synchronous TSG; synchronous TSG generates fewer string pairs than synchronous RF-TAG. By contrast, all these formalisms generate the same weighted string languages and weighted tree languages.

What can we conclude? With statistical parsing, we can use PCFG to get the same SGC as formalisms like probabilistic TSG, and this SGC is beneficial, because training a PCFG from Treebank trees *qua* CFG derivations yields poor results. The richer formalisms provide a conceptual advantage only. But with machine translation, the advantage is real. We cannot use synchronous CFG to get the same SGC as formalisms like synchronous RF-TAG, and therefore we should use the more powerful formalisms if we hope to benefit from their extra SGC. If synchronous CFGs should be

found wanting for machine translation, it would be incorrect to conclude that synchronous grammars in general are not useful for translation; it would also be incorrect to justify this conclusion by reasoning that if PCFGs were good enough for statistical parsing but not machine translation, then more powerful formalisms are not worth considering. Because SGC with respect to string and tree pairs classifies formalisms more finely than SGC with respect to weighted trees does, translation has more to gain by using richer formalisms, and more to lose by ignoring them.

Recent research in syntax-based statistical machine translation has been climbing steadily up the lattice of Figure 4.5b, with TSG being the most recent proposal (Eisner, 2003). Might any of the still higher formalisms be better suited for translation? Moving from synchronous CFG/TSG to synchronous RF-TAG would be a fairly modest extension to a system like that of Yamada and Knight (2001). Their flattening of the training data, which effectively extracts a TSG from it, would be replaced by a TAG extractor like the one described in Section 3.3.1, and the child-reordering operation would be replaced by node-rearranging operations. Thinking of Yamada and Knight's system as a synchronous tree grammar suggests two more extensions. First, it might make more sense to train directly on the observed training data instead of the reconstructed (flattened) training data, as with our parser (Section 3.3.2) and Gildea's TSG system (Gildea, 2003). Second, their flattening step produces only unlexicalized trees and one-level lexicalized trees; it would be interesting to experiment with grammars in which the nontrivial elementary trees had one or more lexical heads.

* * *

We have discussed how to compare grammar formalisms for translation, finding that in this domain formalisms are much more differentiated than with statistical parsing, making the choice of the right formalism more crucial. We observed that recent syntax-based statistical machine translation research has been proposing more and more powerful formalisms, and we ourselves proposed synchronous RF-TAG as a possible next step in the series, showing that it has strictly greater translation power than CFG with the same asymptotic parsing complexity.

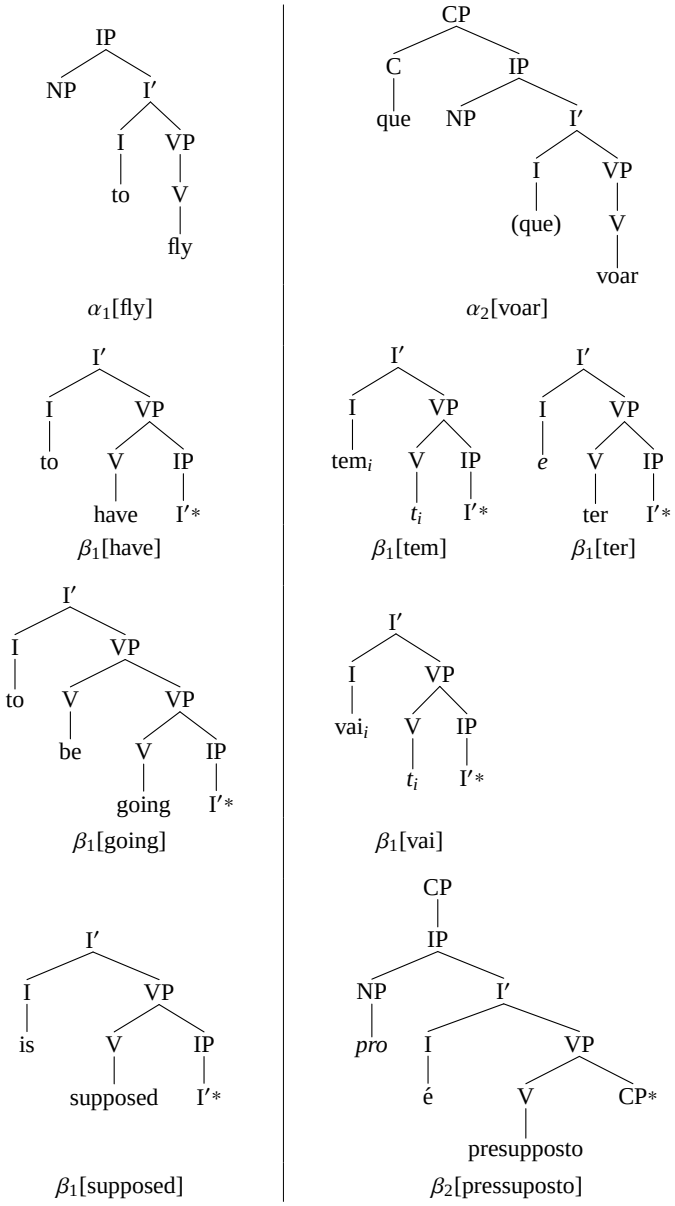


Figure 4.6: Grammar (object-level) for English-Portuguese example.

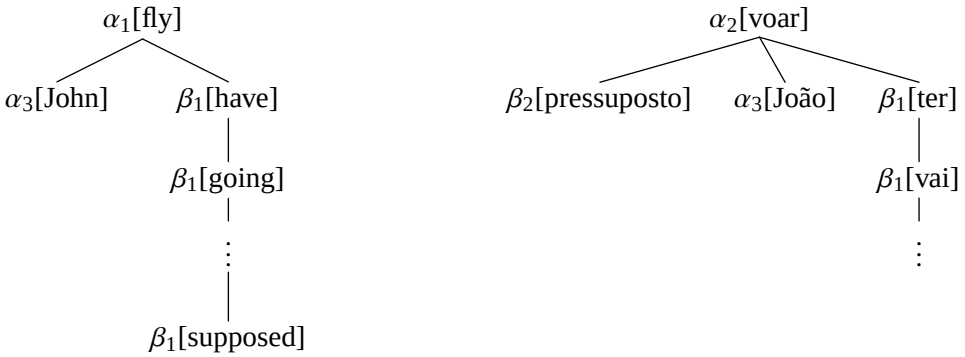
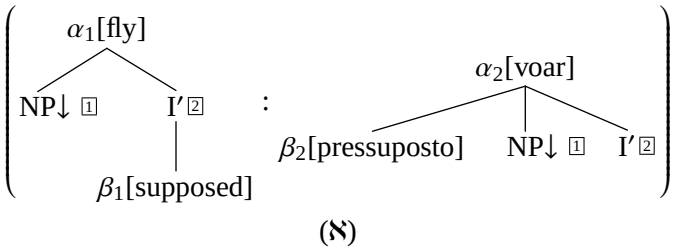
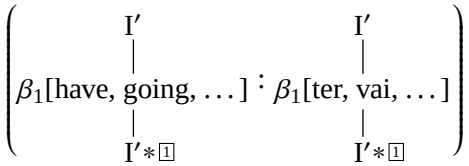


Figure 4.7: Derivation trees (object-level) for English-Portuguese example.



(א)



(ב)

Figure 4.8: Meta-level grammar for English-Portuguese example.

Chapter 5

Applications to biosequence analysis I

A central problem in computational biology is analyzing genetic sequences to determine the structures of the molecules (RNAs, proteins) they code for. Since this problem is analogous to the problem in computational linguistics of describing what structural descriptions are specified by a given utterance, as first observed by Searls (1992), many researchers have tried using formal grammars to analyze biological sequences as well (Sakakibara et al., 1994; Abe and Mamitsuka, 1997; Uemura et al., 1999; Rivas and Eddy, 2000). In this chapter we will attempt a synthesis of this family of approaches, and investigate what properties of formal grammars will determine their success.

5.1 Background

In this section we give an overview of some basic concepts in structural biology from a formal-language-theoretic point of view. A more thorough and conventional introduction can be found in biology textbooks (Branden and Tooze, 1999; Alberts et al., 2002).

5.1.1 Sequences

DNA molecules are built out of nucleotides, which come in four kinds (adenine, thymine, cytosine, and guanine, or A, T, C, and G for short). Each contains a five-carbon sugar, and the only way for two nucleotides to combine is for the fifth (5′) carbon of one to be joined by a covalent bond to

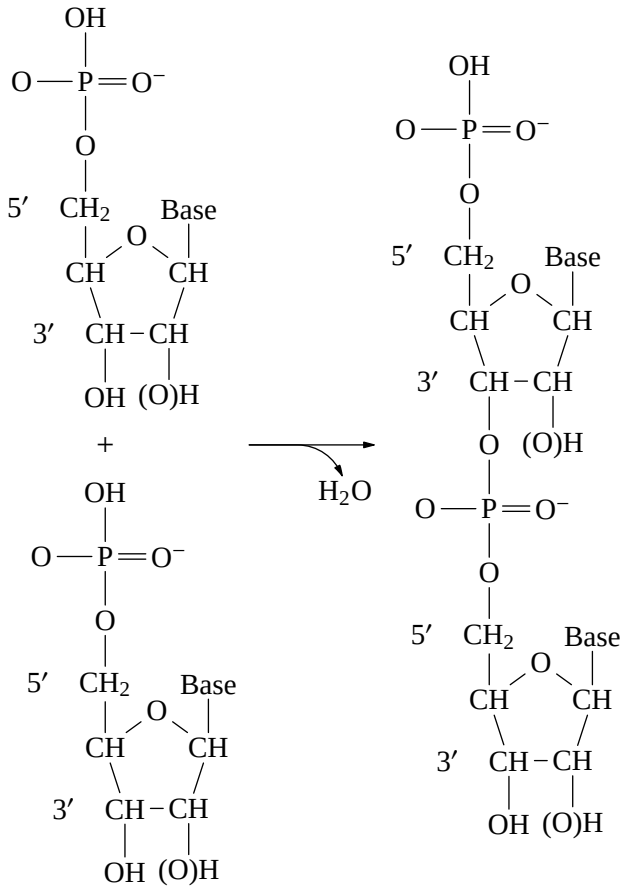


Figure 5.1: Nucleotides combining to form (a segment of) a DNA/RNA molecule. The parenthesized O is absent in DNA.

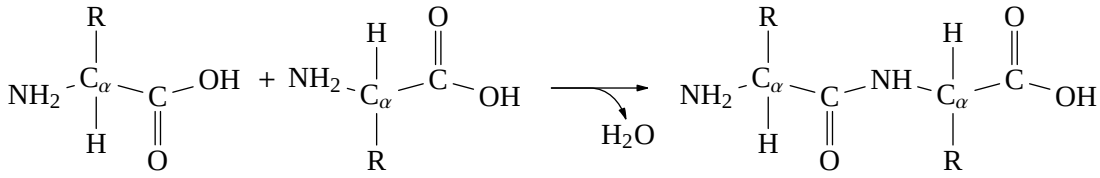


Figure 5.2: Amino acids combining to form (a segment of) a protein. *R* indicates a side chain, which varies from amino acid to amino acid.

A	Ala	alanine	L	Leu	leucine
R	Arg	arginine	K	Lys	lysine
N	Asn	asparagine	M	Met	methionine
D	Asp	aspartic acid	F	Phe	phenylalanine
C	Cys	cysteine	P	Pro	proline
Q	Gln	glutamine	S	Ser	serine
E	Glu	glutamic acid	T	Thr	threonine
G	Gly	glycine	W	Trp	tryptophan
H	His	histidine	Y	Tyr	tyrosine
I	Ile	isoleucine	V	Val	valine

Figure 5.3: Amino acids and their abbreviations.

the third (3′) carbon of the other with a phosphate group in between (see Figure 5.1). This causes nucleotides to form into chains; the end with its 5′ carbon free is called the 5′ end and the end with its 3′ carbon free is called the 3′ end. This asymmetry is significant; the chain is always synthesized starting with the 5′ end, for example. Thus we can think of DNA molecules as strings over $\{a, t, c, g\}$.

DNA is purely informational: it has no function other than to be replicated and transcribed into other alphabets. One such alphabet is that of RNA, which is similar to DNA in structure, except it uses uracil (U) instead of thymine. Thus we can think of RNA molecules as strings over $\{a, u, c, g\}$. RNA is transcribed from DNA by a base-to-base mapping ($A \to U, T \to A, C \to G, G \to C$). RNAs come in various types: transfer RNAs are about 80–100 bases long, and ribosomal RNAs are about 120–2500 bases long.

Proteins are built out of amino acids, which come in twenty kinds (see Figure 5.3). Each amino acid contains an amino group (NH_2) and a carboxyl group (COOH), and the only way for two amino acids to combine is for the amino group of one to be joined by a covalent bond to the carboxyl group of the other (see Figure 5.2). Thus in some diagrams the ends of a protein molecule are labeled N (for the nitrogen atom in the amino group) and C (for the carbon atom in the carboxyl group). Thus, as with DNA and RNA, we can think of proteins as strings over the set of amino acids. Proteins are transcribed from DNA via RNA by the so-called genetic code, which maps triples of

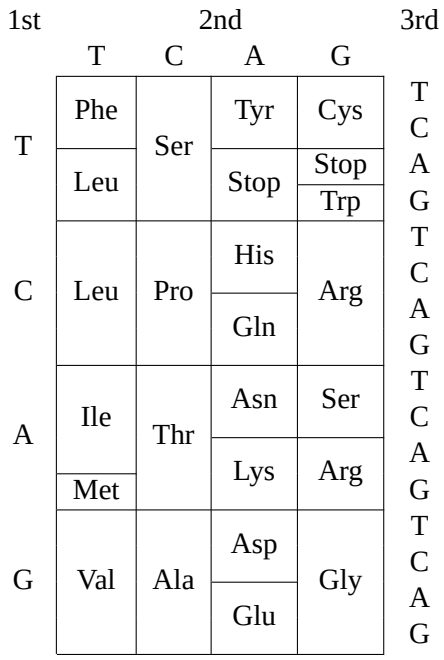


Figure 5.4: The genetic code (DNA → amino acids).

nucleotides (called codons) to amino acids (see Figure 5.4). There are also start and stop codons which permit multiple proteins to be encoded in a single strand of DNA. Proteins can be quite long, exceeding 5000 amino acids in length.

An alternative is to think of these molecules not as strings over an alphabet of nucleotides or amino acids, but as strings of *covalent bonds*. That is, if Σ is the set of nucleotides or amino acids, then we can model a DNA/RNA or protein as a string over $\Sigma \times \Sigma$, and all the strings we would consider would be members of

$$\{\langle a_1, a_2\rangle \cdot \langle a_2, a_3\rangle \cdots \langle a_{n-1} a_n\rangle \mid a_i \in \Sigma\}$$

The advantage of this scheme is that it allows grammars to model structures more flexibly. We will illustrate the use of both schemes below.

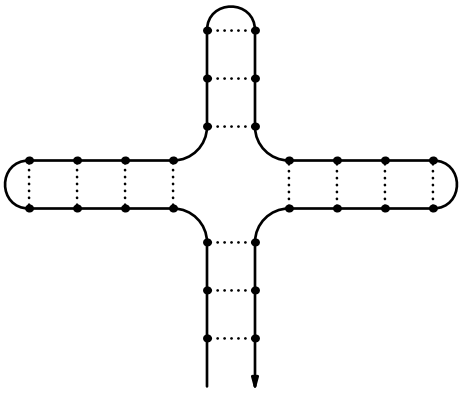


Figure 5.5: Example RNA secondary structure.

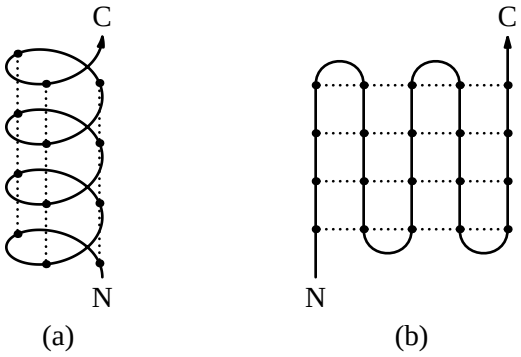


Figure 5.6: Example protein secondary structures: (a) α-helix; (b) β-sheet.

5.1.2 Structures

Though RNA molecules and protein molecules have a linear structure as described above, they do not lie straight in space, because the chemical bonds in them are flexible. Nor do they get bent around every which way, because *self-contacts* form between different parts of the molecule to give it a *secondary structure* (see Figure 5.5), the primary structure being the sequence itself. Distant parts of the secondary structure may come into contact to form a *tertiary structure*.

In RNA, bonds form between complementary bases: A with U, C with G.[1] Thus the secondary/tertiary structure of an RNA molecule is dependent on its primary structure. This structure gives the molecule its particular function: transfer RNAs pair codons with amino acids, and ribosomal RNAs form the machinery which assembles amino acids into proteins.

With proteins the secondary structure is again dependent on its primary structure. Amino acids do not have complementary pairs, but do have varying properties that make some pairings more favorable than others. Protein secondary structures are observed to consist mainly of two types of substructure: *α-helices* and *β-sheets*. Other regions are known as *random coil*. In an α-helix a single region is coiled up into a helix (see Figure 5.6a); in a β-sheet several discontiguous regions are stretched flat to form a sheet (Figure 5.6b). These fold up further into a *tertiary structure*. As with RNAs, the structure of a protein determines its particular function. They perform a wide range of functions, from catalyzing biochemical reactions to giving cells their shape.

In this chapter, we assume that a molecule's structure is uniquely specified by a set of self-contacts, that is, pairs of string positions. This representation is commonly used in the structure prediction literature, where it is also known as a *polymer graph* or *contact map*.

[1] Sometimes uracil can pair with other bases besides adenine.

5.2 Measuring sequence-analysis power

There is a long tradition of applying language-processing techniques (for example, hidden Markov models) to genetic sequences, but the use of formal grammars originated with Searls (1992). Consider the following CFG for RNA sequences:

$$\begin{aligned} &\mathrm{S} \rightarrow \mathrm{Z} \\ &\mathrm{X} \rightarrow \mathrm{aZu} \mid \mathrm{uZa} \mid \mathrm{cZg} \mid \mathrm{gZc} \\ &\mathrm{Y} \rightarrow \mathrm{aY} \mid \mathrm{uY} \mid \mathrm{cY} \mid \mathrm{gY} \mid \epsilon \\ &\mathrm{Z} \rightarrow \mathrm{YXZ} \mid \mathrm{Y} \end{aligned} \tag{5.1}$$

A derivation of a string *w*, represented as a tree (see Figure 5.7), has the same shape as a secondary structure of *w*, because the grammar is written so that only complementary bases appear in the same rule, and CFG derivation trees have the convenient property that symbols from the same rule appear next to each other in the tree. Formal locality corresponds to spatial locality.

It would seem that more complex formalisms do not have this property. For example, in a TAG the adjunction operation can cause parts of an elementary tree to be stretched arbitrarily far apart.[2] But if we distinguish between spatial locality in our drawings of derivations and spatial locality in real molecules, then it becomes apparent that the former is convenient but not crucial. Even if formal locality cannot correspond to spatial locality in our drawings of derivations, they can still correspond to spatial locality in real molecules. In other words, derivations can still describe molecules even if their drawings don't look very much like them.

This gives us the following locality constraint: two nonadjacent monomers can contact only if their corresponding symbols were generated in the same derivation step. All uses of formal grammars to model biological molecules that we know of are based on this principle, though with variations and sometimes only implicitly. In Searls' original treatment (Searls, 1992) and that of

[2]However, Rogers (2003) explores the use of three-dimensional trees to represent derivations of tree-adjoining grammars, and higher-dimensional trees for still more complex formalisms. In a tree-adjoining grammar defined on three-dimensional trees, there is no stretching.

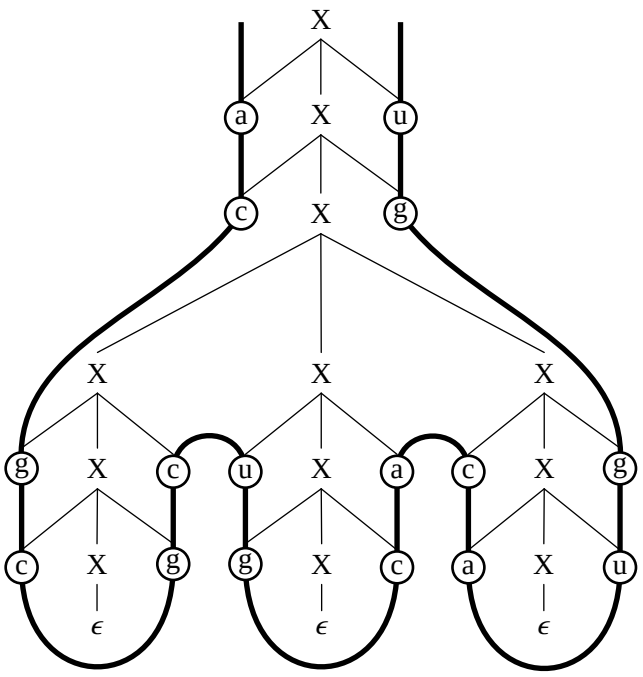


Figure 5.7: Example CFG derivation of RNA secondary structure, with superimposed primary structure. Nonterminal symbols other than X are suppressed for clarity.

Uemura et al. (1999), any two bases appearing in the same right-hand side are assumed to be in contact. Rivas and Eddy (2000) use diagrams reminiscent of Joshi's links (Joshi, 1985). It is a deficiency of the model of Abe and Mamitsuka (1997) that they do not specify self-contacts on their elementary structures, with the result that a single derivation can correspond to multiple structures. Chen and Dill (1998) do not use a grammar at all, but their model can be recast as a CFG (Chiang and Joshi, 2002). They implicitly use an alphabet of covalent bonds (see Section 5.1.1) and specify self-contacts on certain rules. Below is a simplified version of their grammar:

$$
\begin{aligned}
&\mathrm{S} \rightarrow \mathrm{Z} \mid \mathrm{X} \\
&\mathrm{X} \rightarrow \bullet\underset{\cdots}{\mathrm{Z}}\bullet \\
&\mathrm{Y} \rightarrow -\mathrm{Y} \mid - \\
&\mathrm{Z} \rightarrow \mathrm{YX} \mid \mathrm{XZ} \mid \mathrm{YXZ} \mid \mathrm{Y}
\end{aligned}
\tag{5.2}
$$

where $-$ is a terminal symbol (representing a covalent bond) and $\bullet$ is not a symbol at all, but a placeholder (representing a monomer) for a link. The advantage of using an alphabet of covalent bonds here is that multiple rules can create links on more than one monomer. Indeed, this grammar can generate an arbitrary number of links on a single monomer. In our experiments in Chapter 6, we use a grammar of this type; however, since grammars like (5.1) are more intuitive, we will use it as the basis for our examples below.

The grammars used for modeling molecules typically generate the language Σ^*, since we are generally not interested in accepting or rejecting molecules as "grammatical," but determining their structures. Weak generative capacity, then, is not useful as an indicator of the usefulness of a grammar formalism for modeling molecules. What matters is a formalism's derivational generative capacity (1.2), which, as we have already seen, is a measure of ability to generate linked strings. (A slightly different measure would be needed for grammars with alphabets of covalent bonds. The exact relationship between the two should be studied further.)

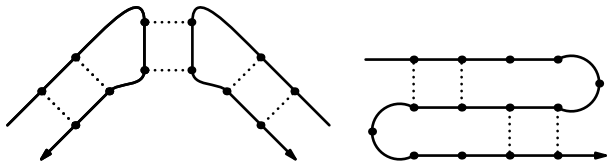


Figure 5.8: Example RNA tertiary structures: (a) kissing hairpins; (b) pseudoknot.

5.3 Linked grammars for sequence analysis

In the previous section we showed how CFG has been used to model RNA and protein structures. Does CFG have enough DGC? It is commonly said that CFG cannot generate crossing dependencies. This is not strictly true, since even a single production can have crossing dependencies:

(5.3) $$X \rightarrow a\ \overset{\frown}{b\ c}\ d$$

Nevertheless there are patterns of crossing self-contacts which occur in nature which are provably not generable by CFG (including Chen and Dill's model). For example, helices involve unbounded series of short crossing self-contacts. RNA tertiary structures involve long-distance crossing self-contacts, for example, *kissing hairpins* and *pseudoknots* (Figure 5.8). Finally, protein β-sheets involve patterns of crossing self-contacts that are well beyond the power of CFG.

5.3.1 Squeezing DGC

We first consider the use of grammar formalisms coverable by CFG which have greater DGC than CFG.

Alpha-helices

Helices have crossing self-contacts of a very limited type. Below we will be interested in helices in which the $(2i)$th monomer is in contact with the $(2i-3)$rd monomer, as this is how helices appear on

a square lattice (see Figure 5.9a). As an exercise, we may generate such helices with the following multicomponent CFG (in the sLMG notation of Section 2.1.2):

$$\begin{aligned} \mathrm{S}(\underbrace{a_1 a_2 y a_3}\, x) &\text{ :– } \mathrm{H}(y : x) \quad a_i \in \Sigma \\ \mathrm{H}(a_1 : \underbrace{y a_2}\, x) &\text{ :– } \mathrm{H}(y : x) \quad a_i \in \Sigma \\ \mathrm{H}(a, \epsilon) & \qquad\qquad a \in \Sigma \end{aligned} \tag{5.4}$$

Since this grammar is component-local, it can be covered by a CFG (indeed, by a right-linear CFG or finite-state automaton). Its derivations can be faithfully represented as derivations of the dissolved grammar (for example, see Figure 5.9b).

Proposition 9. *The linked language L generated by the above component-local multicomponent CFG cannot be generated by any CFG.*

Proof. Suppose L is generated by a CFG G. If a linked string generated by a CFG contains two crossing dependencies as in (5.3), all four terminal symbols involved must come from the same production. By induction, this means that all the terminals in every string of L must come from the same production. Since L is infinite, G must be infinite, which is a contradiction. □

In a previous paper (Chiang, 2002) we showed that component-local multicomponent CFG can even generate linked languages that TAG cannot.

Other kinds of helices can be modeled as well; in particular, if we use an alphabet of covalent bonds, we can model helices in which the ith monomer is in contact with the $(i - 4)$th monomer, as in real α-helices. All of this is somewhat academic, however, because it is easier just to write down the cover grammar directly. For example, the Zimm-Bragg model (Zimm and Bragg, 1959) is a standard theory of the helix-coil transition and can be thought of as a Markov chain.

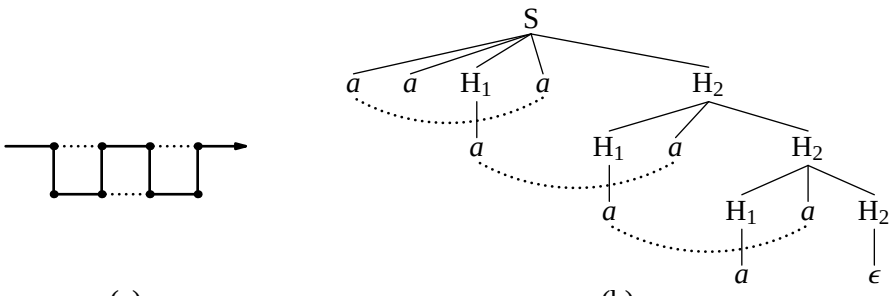


Figure 5.9: (a) α-helix as modeled by square lattice; (b) α-helix represented by derivation of grammar (5.4).

Limited RNA tertiary interactions

The following results show how regular form TAG (see Section 2.1.1) can be used to model limited tertiary interactions in RNAs.

Lemma 10 (linked pumping lemma). *Let L be a linked string set generated by a CFG (or component-local multicomponent CFG). Then there is a constant n such that if $\langle z; \sim_z \rangle$ is in L and $|z| \geq n$, then z may be rewritten as $uvwxy$, with $|vx| > 0$ and $|vwx| \leq n$, such that for all $i \geq 1$, there is a relation $\sim_z^i$ such that $\langle uv^i wx^i y; \sim_z^i \rangle$ is in L and $\sim_z^i$ does not relate any positions in w to any positions in u or y.*

Proof. The proof is analogous to that of the standard pumping lemma (Hopcroft and Ullman, 1979, pp. 125–127). However, a CFG cannot be put into Chomsky normal form without changing its DGC, so we let $n = m^k$ instead of 2^k, where k is the size of the nonterminal alphabet and m is the longest right-hand side or the number 2, whichever is greater. The key difference from the standard proof is the observation that since, for each i, the derivation of $uv^i wx^i y$ can be written as

$$S \stackrel{*}{\Rightarrow} uAy \stackrel{*}{\Rightarrow} uv^i Ax^i y \stackrel{*}{\Rightarrow} uv^i wx^i y$$

for some nonterminal A, no position in w can be contributed by the same derivation step as any

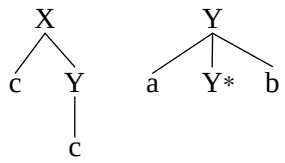
position in u or y.

The generalization to the multicomponent case is straightforward, since a component-local multicomponent CFG G can be converted into a CFG G' which generates the same trees. G' will not generate the same linked strings as G; nevertheless, the equivalence relations generated by G can only relate terminal instances which are first cousins in the derived tree, so for $i \geq 1$, it remains the case that no position in w is related to any position in u or y. □

Proposition 11. *The following linked string set is generable by an RF-TAG but not by any CFG, nor indeed by any component-local multicomponent CFG:*

$$L = \left\{ \text{c a a} \cdots \text{a c b} \cdots \text{b b} \right\}$$

Proof. The following RF-TAG generates L:



But suppose L is generated by some CFG G. For any n given by the linked pumping lemma, let $z = \text{ca}^n\text{cb}^n\text{c}$ satisfy the conditions of the pumping lemma. It must be the case that v and x contain only a's and b's, respectively, or else $uv^iwx^iy \notin L_1$. But then u, w, and y would each have to contain one of the c's, and since the c's are all related, this contradicts the pumping lemma. □

Grammars of this type could be used for structures in which all but a bounded number of self-contacts are nested. For example, in a cloverleaf structure (Figure 5.5), the hairpins may "kiss" (Figure 5.8a), forming a small number of self-contacts crossing over an unbounded number of nested self-contacts. If the number of such self-contacts is indeed bounded, we can write an RF-TAG similar to the one above to generate them (see Figure 5.10).

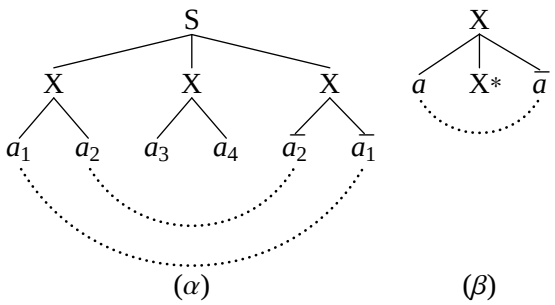


Figure 5.10: RF-TAG for cloverleaf with kissing hairpins. The initial tree α generates the loops (here fixed to two monomers each), and β generates the stem regions.

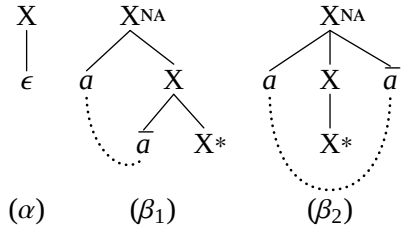


Figure 5.11: TAG fragment for pseudoknots, adapted from the grammar of Uemura et al. (1999)

5.3.2 Beyond CFG

We now examine some existing attempts to apply formalisms beyond CFG to more complex structures.

Pseudoknots

Uemura et al. (1999) use a grammar similar to the one shown in Figure 5.11 to generate pseudoknot structures. A pseudoknot is generated by repeatedly adjoining β_1 into α, then repeatedly adjoining β_2 into the result. Their grammar belongs to an $O(n^5)$-time parseable subclass of TAG, which has been conjectured (Kato, Seki, and Kasami, 2004) to be equivalent to the TAG restriction of Satta and Schuler (1998).

Rivas and Eddy (2000) define a formalism called crossed-interaction grammar. Their definition

$$\begin{aligned}
\mathrm{W}(x_1 y_1 x_2 y_2) &\coloneq \mathrm{W_H}(x_1, x_2), \mathrm{W_H}(y_1, y_2) \\
\mathrm{W_H}(\overset{\cdot\cdot\cdot\cdot}{a}x : y\overset{\cdot\cdot\cdot\cdot}{\bar{a}}) &\coloneq \mathrm{W_H}(x, y) && a \in \Sigma \\
\mathrm{W_H}(\epsilon, \epsilon) & \\
\mathrm{W_H}(x_1 y_1 x_2, y_2) &\coloneq \mathrm{W_H}(x_1, x2), \mathrm{W_H}(y_1, h_2)
\end{aligned}$$

Figure 5.12: sLMG fragment for pseudoknots, adapted from the grammar used by Rivas and Eddy (2000), as a linear sLMG.

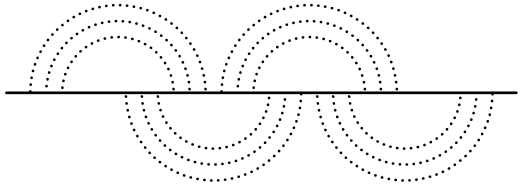


Figure 5.13: Chain of four hairpins beyond the power of TAG.

of this formalism is unclear in many places and seems to be as powerful as type-0 grammars; but a little exegesis shows that what they had in mind is something equivalent to linear sLMG. They then use a grammar similar to the one shown in Figure 5.12 to generate pseudoknot structures. The first three rules generate basic pseudoknots just as the TAG of Figure 5.11. Like a TAG, Rivas and Eddy's grammar has a maximum arity of two and a maximum branching factor of two and is therefore parsable in $O(n^6)$ time, but it lies outside the power of TAG. For example, the last rule allows arbitrary-length chains of hairpins to be generated (Figure 5.13 shows a chain of four), whereas TAG can only build such chains of up to three hairpins.

Beta-sheets

Abe and Mamitsuka (1997) use a formalism called ranked node-rewriting grammar (RNRG) to generate β-sheets. RNRG is essentially TAG with multiple foot nodes on elementary trees; for the present discussion, it is enough to note that RNRG with a branching factor of one is equivalent to

$$\mathrm{S}(a_1x_1b_1, a_2x_2b_2, a_3x_3b_3, a_4x_4b_4) \coloneq \mathrm{S}(x_1, x_2, x_3, x_4) \qquad a_i, b_i \in \Sigma$$
$$\mathrm{S}(\epsilon, \epsilon, \epsilon, \epsilon)$$

Figure 5.14: sLMG fragment for β-sheet, adapted from one of the grammars of Abe and Mamitsuka (1997).

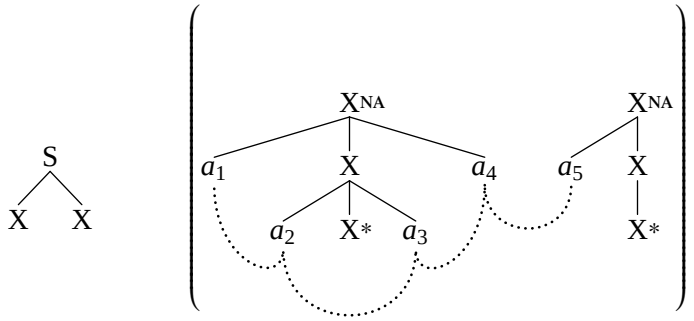


Figure 5.15: Set-local multicomponent TAG for protein β-sheet of Figure 5.6b.

linear sLMG with a maximum branching factor of one. Figure 5.14 shows a grammar for generating a β-sheet of eight alternating strands. Set-local multicomponent TAG offers a similar solution; Figure 5.15 shows a grammar to generate the five-strand sheet of Figure 5.6b.

The difficulty is that parsing of these grammars is exponential in the number of strands per sheet. Moreover, every grammar imposes some upper bound, so that there is no single grammar that can generate all β-sheets. For this reason, approaches of this type appear to be prohibitively expensive.

A second problem is that this analysis is susceptible to a kind of spurious ambiguity in which a single structure can be derived in multiple ways. For example, consider Figure 5.16. In order to generate the β-sheet (a), we need trees like (b) and (c). But either of these trees can be used by itself to generate the β-sheet (d). The grammar must make room for the maximum number of strands, but when it does not use all of it, ambiguity can arise. It should be possible to carefully write the grammar to avoid much of this ambiguity, but we have not been able to eliminate all of it even for the single-component TAG case.

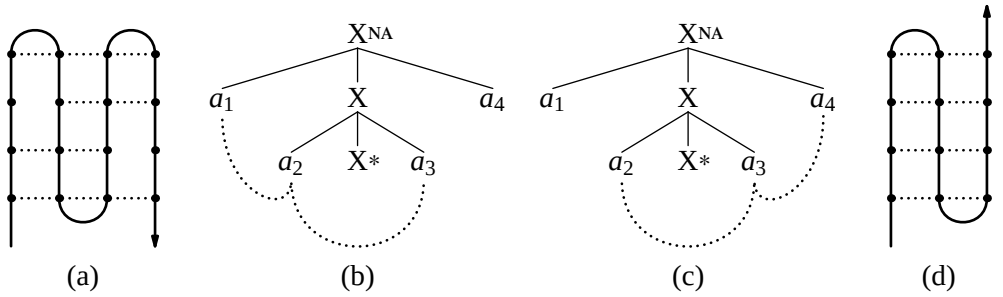


Figure 5.16: Illustration of spurious ambiguity in a multicomponent TAG.

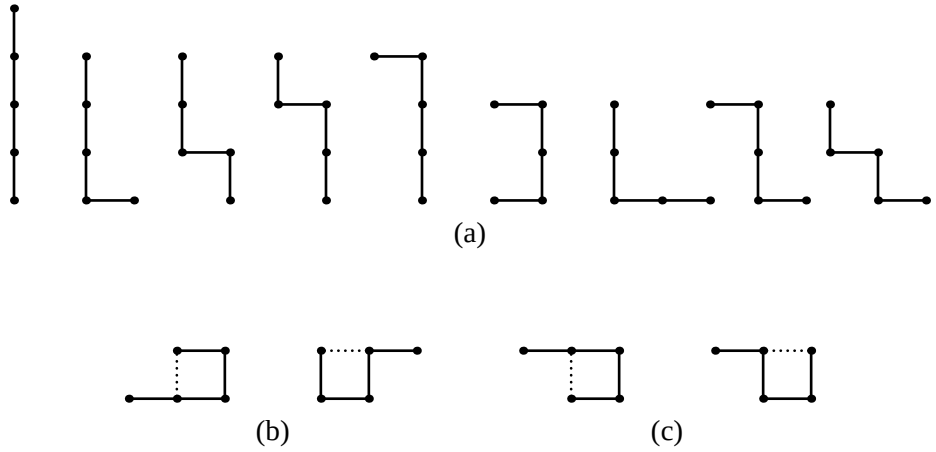


Figure 5.17: All possible conformations of a 5-mer on a square lattice (modulo rotational and reflectional symmetry), grouped according to structures, (a), (b), and (c).

5.4 Computing probabilities and partition functions

Just as with natural language parsing, a grammar can assign many derivations to a single string, and we would like some measure of the relative importance of each of them—for example, a probability distribution. Attempts have been made to estimate probabilities from databases (Sakakibara et al., 1994; Abe and Mamitsuka, 1997). However, maximizing the likelihood of a database would not seem to make much sense, since the database does not represent a uniform sample of any naturally occurring distribution (as a text corpus arguably does). Maximizing the conditional likelihood might be a sounder approach (Hockenmaier, p.c.). But physical theory provides a more principled

alternative. If we have some way of calculating the energies of structures, minimizing the energy will give us the most stable structure. In fact, statistical physics can tell us the full probability distribution over structures in terms of their energy.

Consider a set of many identical molecules, each in a particular *conformation*, or arrangement in physical space. In the HP lattice model (Lau and Dill, 1989), conformations are represented by self-avoiding walks on a lattice (see Figure 5.17). Note that there can be more than one conformation corresponding to a single structure; for example, in Figure 5.17, all the conformations in the first row have the same structure. The maximum-entropy probability distribution over the conformations of these molecules, subject to the constraint that their total energy must be constant, is known as the *Boltzmann distribution*:

$$P_j = \frac{e^{-E_j/kT}}{Q} \tag{5.5}$$

where j ranges over conformations, E_j is the energy of conformation j, T is the temperature, k is Boltzmann's constant, and

$$Q = \sum_j e^{-E_j/kT} \tag{5.6}$$

is known as the *partition function*. Since we are more interested in distributions over structures than conformations, we may regroup both the Boltzmann distribution and the partition function according to structure:

$$P_j = \frac{\Omega_j e^{-E_j/kT}}{Q} \tag{5.7}$$

$$Q = \sum_j \Omega_j e^{-E_j/kT} \tag{5.8}$$

where j now ranges over structures, and Ω_j is the number of conformations with structure j.

When T is low, the Boltzmann distribution says that the most probable structure will simply be

the one(s) with the lowest energy: let $E_{min} = \min E_j$, then

$$\lim_{T\to 0} P(j) = \lim_{T\to 0} \frac{\Omega_j e^{-E_j/kT}}{\sum_{j'} \Omega_{j'} e^{-E_j/kT}} \tag{5.9}$$

$$= \lim_{T\to 0} \frac{\Omega_j e^{-E_j/kT+E_{min}/kT}}{\sum_{j'} \Omega_{j'} e^{-E_j/kT+E_{min}/kT}} \tag{5.10}$$

$$= \begin{cases} \frac{\Omega_j}{\sum_{j' \text{ s.t. } E_{j'} = E_{min}} \Omega_{j'}} & \text{if } E_j = E_{min}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.11}$$

When T is high, the Boltzmann distribution predicts a uniform distribution of conformations, therefore giving preference to structures with more conformations.

The partition function gives the *effective accessibility* of each conformation (0 being fully inaccessible, 1 being fully accessible) and turns out to be the more useful object to compute. All the statistical-mechanical properties of a system, including the Boltzmann distribution, can be computed from it. We can use it, for example, to understand the folding process of a molecule, or the changes it undergoes under varying conditions, which can shed further light on its function.

How do we compute the partition function using a weighted grammar? If we can assign quantities ω_π and ΔE_π to each elementary structure of the grammar π such that for every structure j built out of elementary structures $\pi_1, \ldots, \pi_n$,

$$\prod_i \omega_{\pi_i} \approx \Omega_j \tag{5.12}$$

$$\sum_i \Delta E_{\pi_i} \approx E_j \tag{5.13}$$

and assign the weight $\omega e^{\Delta E/kT}$ to each elementary structure, then the weight of the derivation of j will be

$$\prod_i \omega_{\pi_i} e^{-\Delta E_{\pi_i}/kT} \approx \Omega_j e^{-E_j/kT} \tag{5.14}$$

The problem, then, is to design the grammar such that the energy increments (ΔE_π) and conformation counts (ω_π) can be estimated accurately.

Calculating energies Attaching energies to rules is common practice among previous grammatical approaches to structure prediction. Here we describe a particularly simple energy model. In the HP model (Lau and Dill, 1989), monomers are classified as either hydrophobic (h) or polar (p), and hh contacts are favorable. That is, the energy of an hh contact is $\varepsilon < 0$, and the energy of other self-contacts is zero. Therefore, we can adapt grammar (5.1) as follows, letting $q_{\text{hh}} = e^{-\varepsilon/kT}$:

$$
\begin{aligned}
&\mathrm{S} \xrightarrow{1} \mathrm{Z} \\
&\mathrm{X} \xrightarrow{q_{\text{hh}}} \mathrm{hZh} \\
&\mathrm{X} \xrightarrow{1} \mathrm{hZp} \mid \mathrm{pZh} \mid \mathrm{pZp} \\
&\mathrm{Y} \xrightarrow{1} \mathrm{hY} \mid \mathrm{pY} \mid \epsilon \\
&\mathrm{Z} \xrightarrow{1} \mathrm{YXZ} \mid \mathrm{Y}
\end{aligned}
\tag{5.15}
$$

(We use the factor $q_{\text{hh}} = e^{-\varepsilon/kT}$ here instead of the energy ε itself, so that the weights can be multiplied instead of added, for consistency with the other grammars in this section.) A parser that computes the minimum-weight derivation of a string under this grammar would compute the native structure of the corresponding molecule.

Counting conformations Attaching conformation counts to grammar rules has not been explored previously to our knowledge. Chen and Dill (1995) use a matrix computation to estimate conformation counts in polynomial time by dividing each structure into substructures and ignoring excluded volume between substructures. That is, the substructures are counted separately, and then the counts are multiplied together. The grammar can check for collisions between substructures to the extent that the shape of a substructure can be finitely encoded; but in general collisions between substructures are ignored. Their algorithm may alternatively be viewed as a parser for an implied

grammar (Chiang and Joshi, 2002). We can similarly add conformation counts to grammar (5.15):

$$
\begin{aligned}
\mathrm{S} &\xrightarrow{C(l-1)} \mathrm{Z}^l \\
\mathrm{X} &\xrightarrow{\frac{1}{4}U(l)q_{\mathrm{hh}}} \mathrm{hZ}^l\mathrm{h} \\
\mathrm{X} &\xrightarrow{\frac{1}{4}U(l)} \mathrm{hZ}^l\mathrm{p} \mid \mathrm{pZ}^l\mathrm{h} \mid \mathrm{pZ}^l\mathrm{p} \\
\mathrm{Y}^{l+1} &\xrightarrow{1} \mathrm{hY}^l \mid \mathrm{pY}^l \\
\mathrm{Y}^0 &\xrightarrow{1} \epsilon \\
\mathrm{Z}^{k+l+1} &\xrightarrow{1} \mathrm{Y}^k\mathrm{XZ}^l \\
\mathrm{Z}^l &\xrightarrow{1} \mathrm{Y}^l
\end{aligned}
\tag{5.16}
$$

where the superscripts are counters used for measuring lengths. Y generates open chains and X generates closed loops, and Z generates combinations of the two, counting the latter as having length one. When an X forms a closed loop out of a Z of length l, it multiplies in a conformation count of $\frac{1}{4}U(l)$, where $U(l)$ is the number of neighbor-avoiding loops of length l on the two-dimensional lattice (l even). When S forms the whole molecule out of a Z of length l, it multiplies in a conformation count of $C(l-1)$, where $C(l)$ is the number of neighbor-avoiding walks of length l. For large l we use approximations of the form $A\mu^l l^{\gamma-1}$ (Madras and Slade, 1993). For $U(l)$, we use $A = 1.3$, $\gamma = \frac{11}{32}$; for $C(l)$ we use $A = 0.034$, $\gamma = 0.5$; for both formulas we use $\mu = 2.3$. These values were chosen to approximate the results of exact enumeration; more precise or more theoretically-motivated values would be desirable.

Computing partial sums of the partition function A weighted derivation forest lets us compute the total weight Q of the forest (using the Inside algorithm), or the weight of a single derivation (corresponding to a single structure). However, we may want to further group structures into bins in various ways and compute the total weight of each bin. For example, we might want to group structures according to their energy level and ask what the total contribution of each energy level is. Or we might group structures according to how many self-contacts are present or not present in

the native structure (Chen and Dill, 2000).

To do this, we incorporate the bins into the nonterminal alphabet. For example, to group the derivations of grammar (5.1) by the number of self-contacts, we would write:

$$
\begin{aligned}
\mathrm{S}^{n} &\to \mathrm{Z}^{n} \\
\mathrm{X}^{n+1} &\to \mathrm{hZ}^{n}\mathrm{h} \mid \mathrm{hZ}^{n}\mathrm{p} \mid \mathrm{pZ}^{n}\mathrm{h} \mid \mathrm{pZ}^{n}\mathrm{p} \\
\mathrm{Y}^{0} &\to \mathrm{hY}^{0} \mid \mathrm{pY}^{0} \mid \epsilon \\
\mathrm{Z}^{m+n} &\to \mathrm{Y}^{0}\mathrm{X}^{m}\mathrm{Z}^{n} \\
\mathrm{Z}^{0} &\to \mathrm{Y}^{0}
\end{aligned}
\tag{5.17}
$$

This grammar has multiple start symbols S^{n}, one for each bin. The total weight of S^{n} is the total weight of states with n self-contacts. Using this grammar we can also calculate the mean and variance of the number of self-contacts, or the most likely structure for each number of self-contacts. The rule weights come from grammar (5.16); combining these two grammars, we get:

$$
\begin{aligned}
\mathrm{S}^{n} &\xrightarrow{C(l-1)} \mathrm{Z}^{l,n} \\
\mathrm{Y}^{l+1,0} &\xrightarrow{1} \mathrm{hY}^{l,0} \mid \mathrm{pY}^{l,0} \\
\mathrm{Y}^{0,0} &\xrightarrow{1} \epsilon \\
\mathrm{Z}^{k+l+1,m+n} &\xrightarrow{1} \mathrm{Y}^{k,0}\mathrm{X}^{m}\mathrm{Z}^{l,n} \\
\mathrm{Z}^{l,0} &\xrightarrow{1} \mathrm{Y}^{l,0} \\
\mathrm{X}^{n+1} &\xrightarrow{\frac{1}{4}U(l)q_{\mathrm{hh}}} \mathrm{hZ}^{l,n}\mathrm{h} \\
\mathrm{X}^{n+1} &\xrightarrow{\frac{1}{4}U(l)} \mathrm{hZ}^{l,n}\mathrm{p} \mid \mathrm{pZ}^{l,n}\mathrm{h} \mid \mathrm{pZ}^{l,n}\mathrm{p}
\end{aligned}
\tag{5.18}
$$

Parsing CFG typically has time complexity $O(|G|n^3)$. The fourth rule schema above is the one which has the most instantiations: $O(n^2B^2)$, where B is the number of bins, for two length indices and two bin indices. However, l in this rule does not contribute to parsing complexity because it is always equal to the width of the span of the Y; therefore this grammar can be parsed in time

$O(n^4B^2)$. Note that B is the number of bins *for a given string*, which needs to be finite. If the bins are energy levels, they are all linear combinations (with integer coefficients) of the ΔE's, which are drawn from a fixed, finite set. If there is a number x such that each ΔE can be expressed as an integer multiple of x (it suffices for the ΔE's to be all rational), then B will be linear in the number of self-contacts. If the number of self-contacts a single terminal can participate in is bounded (in this case, it is bounded to one), then $B \in O(n)$, giving an overall complexity of $O(n^6)$. Similar reasoning fixes asymptotic upper bounds on other binning schemes as well.

Implementation details In practice it is more efficient to calculate the partition function (including energies, lengths, counts, and bins) offline, after discarding chart items which are not part of a complete derivation. Since the nonterminal indices for lengths and bins do not affect grammaticality, we can first parse with grammar (5.1) and then reparse only the resulting forest with grammar (5.18).

* * *

In this chapter we have provided a synthesis of current research in the application of formal grammars to biological sequence analysis. We have characterized the ability of grammar formalisms to model secondary/tertiary structures by their DGC, and introduced a few novel ways of using extra DGC to model more complex structures. Finally, we have shown how to use extended weights in a grammar to compute partition functions, thus reformulating Chen and Dill's nongrammatical model as a weighted CFG. In the next chapter we explore the use of the technique of *intersection* to extend this model to more complex structures like bundles of α-helices, or possibly β-sheets.

Chapter 6

Applications to biosequence analysis II

Another strategy for obtaining more SGC out of a grammar formalism is to combine multiple grammars into a single system which accepts the intersection of the languages accepted by the component grammars, and which assigns to each string the *unification* (in some sense) of the structural descriptions assigned by the component grammars. This technique has not received much attention in computational linguistics, probably because linguistic structures tend to be hierarchical, and it is not very clear how to unify multiple hierarchical structures into a single one. With molecular structures, on the other hand, there is a straightforward way of unifying two linkings of a string: simply form the union of the links. In this chapter we discuss the strengths and weaknesses of several variants of this strategy.

6.1 Intersecting CFLs and CFLs: a critique

In the first style of intersection, a language $L = L_1 \cap L_2$ is simply specified by grammars for L_1 and L_2. A string is recognized as belonging to L just in case it is recognized as belonging to both L_1 and L_2. There is no interaction between the two grammars at the level of their derivations or structural descriptions; the only interaction is that each filters the other's generated strings.

Context-free languages are not closed under intersection (Hopcroft and Ullman, 1979, pp. 134–135). This suggests the possibility of using two or more CFGs to recognize a language beyond the power of CFG. Brown and Wilson (1996) propose just this approach for RNA pseudoknots. They

observe that $\{a^m g^* u^m c^*\}$ and $\{a^* g^n u^* c^n\}$ are context-free languages, but their intersection is the non-context-free language $\{a^m g^n u^m c^n\}$. This language is reminiscent of a set of pseudoknots: the m a's and u's form one hairpin, and the n g's and c's are the other. Therefore this would seem to be an efficient way of modeling pseudoknots.

However, in order for the pseudoknot to be well-formed, the two hairpins must interlock without colliding. That is, the base pairings must cross, but no two pairings should involve the same base. But the only reason the above example achieves this is because one hairpin has only a's and u's and the other has only c's and g's—that is, each symbol indicates overtly which hairpin it belongs to. For real molecules, both component grammars would have to generate at least all possible hairpins, or $\{vww^R x\}$. In that case there would be no way of preventing the component grammars from missing each other or colliding.

Brown and Wilson recognize that there is a problem, but it is not clear whether they appreciate how serious it is. Their solution is to employ a special parsing strategy that uses the results of parsing with the first grammar to constrain the parse with the second; then the string is reparsed with the first, then again with the second. This procedure works only for their pair of grammars and only approximates the desired computation.

The root of the problem is that intersection only operates on strings, not structural descriptions. It allows parallel structural descriptions to be derived independently, then filters them on the basis of their string yields. The above example attempts to harness this filtering to generate only well-formed pseudoknots, but in order to do so it assumes that there is more information in the string languages than there really is.

6.2 Intersecting CFGs and finite-state automata

Suppose, however, that G_1 is a CFG and G_2 is a right-linear CFG, or a grammar that can be covered by a right-linear CFG. Since G_2 generates a regular language and CFLs are closed under intersection with regular languages, it is possible to construct a new CFG $G_\cap$ that generates $L(G_1) \cap L(G_2)$. Though there is no increase in WGC as in the previous section, there is still an increase

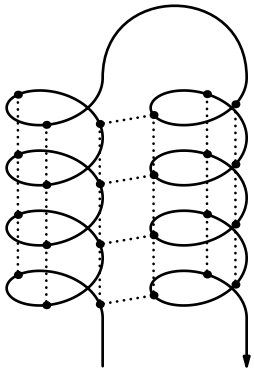


Figure 6.1: Two-helix bundle.

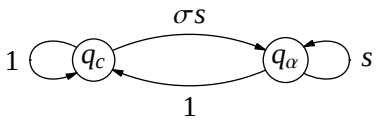


Figure 6.2: The Zimm-Bragg model as a weighted finite-state automaton.

in SGC, because the resulting system assigns to each string the superposition of its structural descriptions assigned by G_1 and G_2.[1] The advantage of this approach is that the two grammars are much easier to control when encapsulated into a single grammar than in the previous section.

In Section 5.3.1 we mentioned the Zimm-Bragg model of α-helices (Zimm and Bragg, 1959), which gives partition functions for conformations with local self-contacts. Chen and Dill's model gives partition functions for conformations with nested nonlocal self-contacts, but thus far it has been impossible to compute partition functions for chain molecules having both local *and* nonlocal interactions, as in bundles of helices (Figure 6.1).

But the Zimm-Bragg model is formally a weighted finite-state automaton: every helix unit preceded by a coil unit has weight σs, and every helix unit preceded by a helix unit has weight s

[1]Of course, $G_\cap$ itself has no more power than an ordinary CFG; it is the combination of G_1 and G_2 that has greater SGC. We leave for future work the question of when this combination can be covered by a CFG.

(Figure 6.2). Moreover, this automaton could be thought of as a cover grammar for a component-local multicomponent CFG for helices (Section 5.3.1). And we have already shown that Chen and Dill's model is equivalent to a weighted CFG. Therefore, we can use the machinery of formal grammars to combine these two models easily. The combined system would generate linked strings representing two-helix bundles, and the constructed weighted CFG would correctly calculate the partition function.

6.2.1 Integrating the Zimm-Bragg model and the HP model

Before integrating the Zimm-Bragg model into Chen and Dill's model, we must first adapt it to the HP lattice model which underlies Chen and Dill's model. This will apply both to our final grammar-based model as well as the exact enumeration we will evaluate it against.

In a real α-helix, for each i, the ith monomer is in contact with the $(i-4)$th monomer, and between the ith and $(i-1)$st monomer, there are two bond angles (like hinges) which must be frozen into the correct shape. The Zimm-Bragg model models the self-contacts by giving each an energy of ε_s, and it models the freezing by giving a conformation count of $r < 1$ to each bond angle, representing the relative lack of conformational freedom of a helix relative to random coil. Therefore the first turn of the helix should get a weight of $r^6 e^{-\varepsilon_s/kT}$. The r^6 factor is for the six frozen bond angles between the first through fourth monomers, and the $e^{-\varepsilon_s/kT}$ for the self-contact between the first and fourth monomers. Then each subsequent monomer, because it freezes two more bond angles and adds one more self-contact, gets a weight of $r^2 e^{-\varepsilon_s/kT}$. The simplest version of the model collapses the whole first turn into the first monomer; thus the first monomer gets a weight of σs, where $\sigma \approx r^4$ and $s \approx r^2 e^{-\varepsilon_s/kT}$, and subsequent monomers get a weight of s.

Now on a square lattice, a helix is modeled as shown in Figure 5.9a. This is not completely accurate (cf. Figure 5.6a): only every other monomer creates a new self-contact, and the $(2i)$th monomer is in contact with the $(2i-3)$rd monomer. Nevertheless, we still give every monomer an energy of ε_s, plus an energy of ε_{hh} for every hh contact.

Since we explicitly count conformations on a lattice, we are already counting random coil as

having more conformations than helix—by a factor of approximately μ (the connective constant from Section 5.4) per monomer. Ideally, then, the factor r would be superfluous. In that case we would simply give each helix unit a weight of $s = e^{-\varepsilon_s/kT}$. However, because the lattice model does not match reality very exactly, we keep a correction factor on s: $s = s_0 e^{-\varepsilon_s/kT}$, where $s_0 \approx \mu r^2$. Moreover, in the lattice model the first turn is no more difficult to form than subsequent turns. Therefore we must retain the σ parameter as well.

To summarize, the factors contributing the weight of a conformation are:

$q_{\text{hh}} = e^{-\varepsilon_{\text{hh}}/kT}$	for every hh contact
$s = s_0 e^{-\varepsilon_s/kT}$	for every non-initial monomer in a helix
σs	for the first monomer in a helix

6.2.2 Intersecting the grammars

For the two-helix bundle problem, our grammar is the CFG (5.2), and our finite-state automaton is shown in Figure 6.3. It is more complicated than the Zimm-Bragg model because it tries to model the shape of a helix in a square lattice. Like the Zimm-Bragg model, it does not explicitly generate self-contacts; but it can be viewed as a cover grammar for a grammar which does (see Section 5.3.1).

The state q_c is for coil; the states q_{ij} are for helices, where i cycles through four values corresponding to the periodicity of the shape of a helix in a square lattice (see Figure 5.9a), and j is used to ensure that all helices are at least six monomers long.

The procedure for intersecting a context-free grammar with a finite-state automaton is due to Bar-Hillel et. al (1964, pp. 149–150). Given a CFG $G = \langle V, \Sigma, S, P \rangle$ and a finite-state automaton (without ϵ-transitions) $M = \langle \Sigma, Q, Q_0, Q_f, \delta \rangle$, the new CFG G' has nonterminal alphabet $Q \times V \times Q \cup S'$, where S' is a new start symbol not in V; and its production set consists of all productions

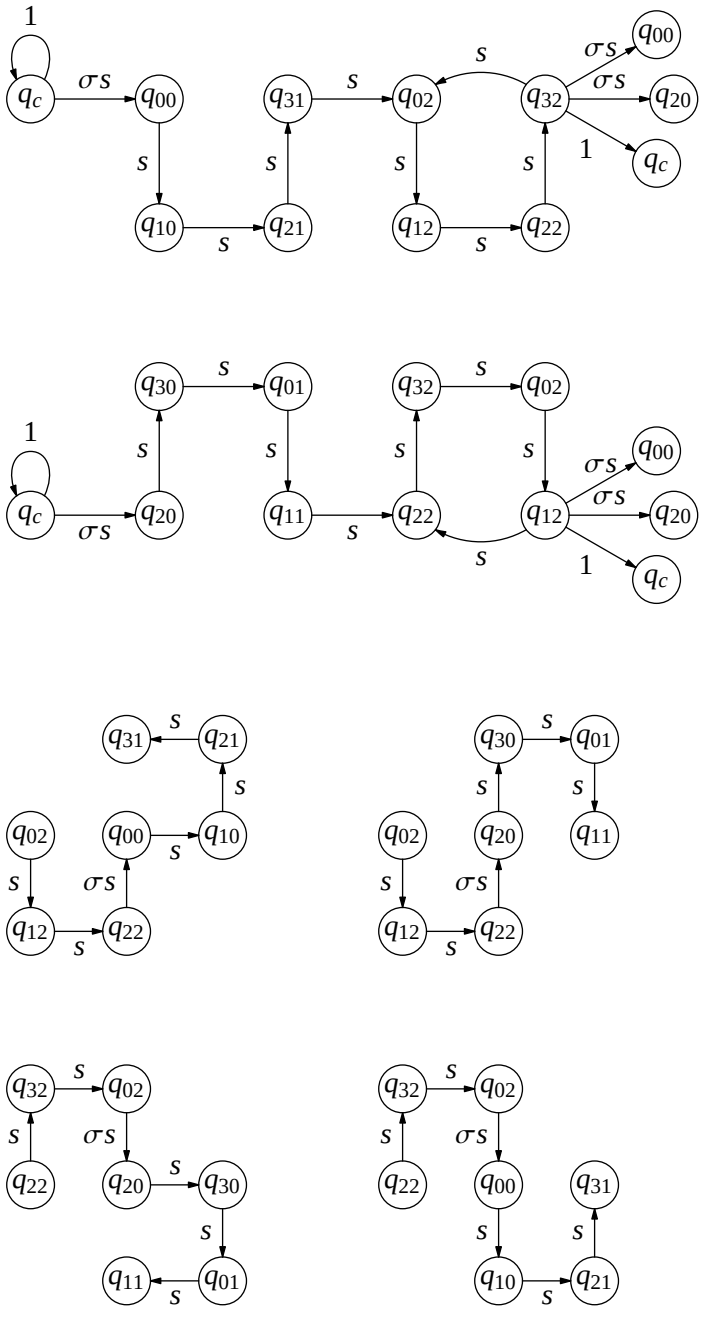


Figure 6.3: Automaton for helices in a square lattice. Nodes with the same label represent the same state; the state is shown in multiple locations for visual clarity. The full automaton has the union of the transitions shown in all six diagrams. The initial states are $\{q_c, q_{00}, q_{20}\}$; the final states are $\{q_c, q_{12}, q_{32}\}$.

of the form

$$\langle q_0, X, q_n \rangle \to \langle q_0, \alpha_1, q_1 \rangle \langle q_1, \alpha_2, q_2 \rangle \cdots \langle q_{n-1}, \alpha_n, q_n \rangle$$

where $X \to \alpha_1 \cdots \alpha_n \in P$ ($n > 0$), and for each i, either $\alpha_i \in V$ or else $\alpha_i \in \Sigma$ and $\langle q_{i-1}, \alpha_i, q_i \rangle \in \delta$; and

$$\langle q, X, q \rangle \to \epsilon$$

for all $q \in Q$, where $X \to \epsilon \in P$; and, finally,

$$S' \to \langle q_0, S, q_f \rangle$$

for all $q_0 \in Q_0$, $q_f \in Q_f$. The resulting grammar generates the language $L(G) \cap L(M)$.

The difference between this construction and one like Brown and Wilson's for pseudoknots is that the two component grammars are fully integrated, so that we may let them control each other however we please. For example, when two helices come together to form a bundle, self-contacts should only be allowed between monomers on the sides of the helices facing each other. Our original CFG had the rule $X \to Z$ which generated a self-contact; in the intersected grammar its corresponding productions include those of the form $\langle q_{ij}, X, q_{i'j'} \rangle \to \langle q_{ij}, Z, q_{i'j'} \rangle$. We may now stipulate that $i, i' \in \{0, 2\}$ for such productions, which ensures that only one side of a helix may participate in nonlocal contacts.

6.2.3 Computing the partition function

We compute the partition function offline as described in Section 5.4, with some modifications. First, we incorporate the weights σ and s from the Zimm-Bragg model, and an additional factor q_{hh} for every helical hh contact.

Second, previously we estimated the number of conformations of a loop of length l as $\frac{1}{4}U(l)$ and the number of conformations of the tails with combined length l as $C(l-1)$. Now that these loops and tails may include helices, which are rigid, we must adjust these estimates. Our current

approach is simply to count each helix as a single step in a neighbor-avoiding walk, without trying to take into account the length of the helix.

Finally, since the grammar is most error-prone with closed conformations, we use a special set of rules for loops of length eight or less, shown below (weights are conformation counts only):

$$
\begin{aligned}
&\mathrm{X} \xrightarrow{1} \mathrm{HHHX'HHH} \\
&\mathrm{X} \xrightarrow{1} \mathrm{UX'HHH} \\
&\mathrm{X} \xrightarrow{1} \mathrm{HHHX'U} \\
&\mathrm{X} \xrightarrow{1} \mathrm{HXH} \\
&\mathrm{X} \xrightarrow{1} \mathrm{X'X'X'} \\
&\mathrm{X} \xrightarrow{4} \mathrm{CCCCCCC} \\
&\mathrm{X'} \rightarrow \mathrm{X} \mid \mathrm{C} \\
&\mathrm{U} \rightarrow \mathrm{H} \mid \mathrm{C} \\
&\mathrm{H} \rightarrow \langle q, a, q' \rangle \qquad \langle q, a, q' \rangle \in \delta_H \\
&\mathrm{C} \rightarrow \langle q, a, q' \rangle \qquad \langle q, a, q' \rangle \in \delta_C
\end{aligned}
\tag{6.1}
$$

These rules do not exhaustively cover all possible loops of length eight or less; a number of possibilities were left out somewhat arbitrarily to limit overcounting. Chen and Dill's steric compatibility matrices might be a more principled solution.

6.2.4 Evaluation against exact enumeration

We compared our parser against exact enumeration in various ways. First, we tried the sequence hpphhpphhpphhpphhpph, which has minimum-energy structures high in both helix units and hh contacts (Figure 6.4a). In this experiment and those below, $\sigma = 0.01$. Figure 6.5 shows the average number of helix units as a function of the parameters s and q_{hh}, and Figure 6.6 shows superimposed cross-sections of these functions; the output of the parser qualitatively agrees with that of the exact

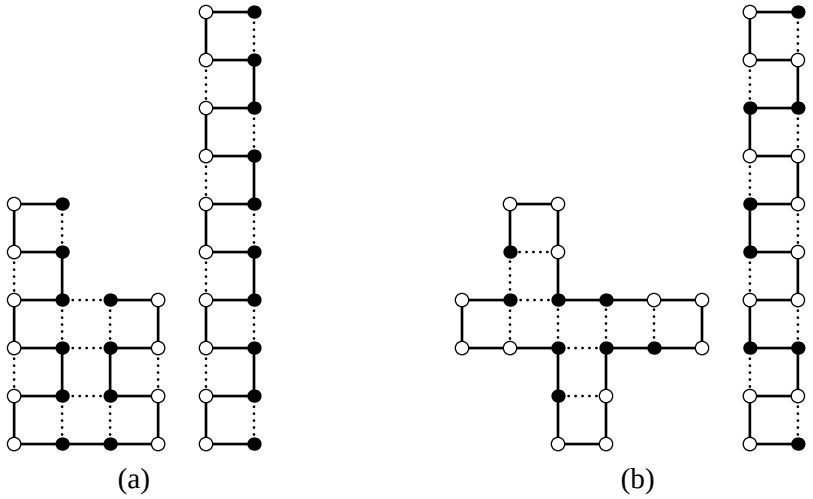


Figure 6.4: Some favorable structures for example sequences: (a) hpphhpphhpphhpphhpph; (b) hppphhpppphhpppphhppph. Hydrophobic monomers (h) are indicated by black circles; polar monomers (p) by white circles.

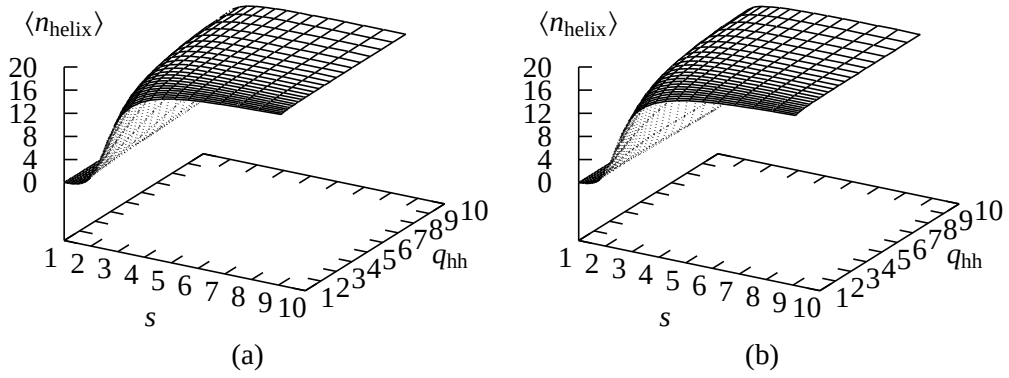


Figure 6.5: Comparison against exact enumeration. Sequence: hpphhpphhpphhpphhpph; helix units versus s and q_{hh}. (a) Exact enumeration. (b) Parser.

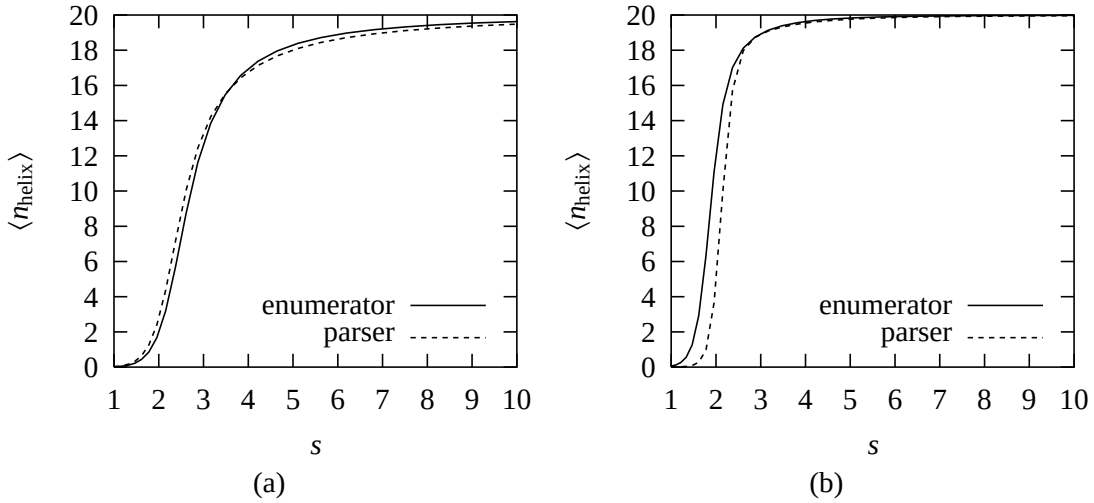


Figure 6.6: Comparison against exact enumeration. Sequence: hpphhpphhpphhpphhpph; helix units versus s. (a) $q_{\mathrm{hh}} = 1$. (b) $q_{\mathrm{hh}} = 10$.

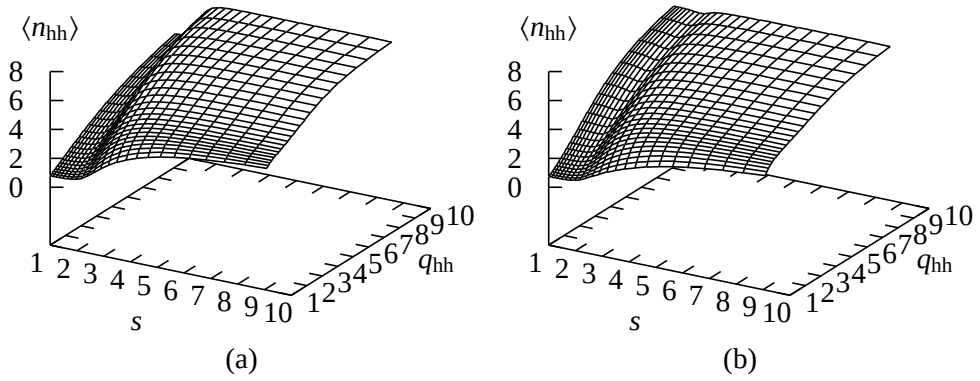


Figure 6.7: Comparison against exact enumeration. Sequence: hpphhpphhpphhpphhpph; hh contacts versus s and q_{hh}. (a) Exact enumeration. (b) Parser.

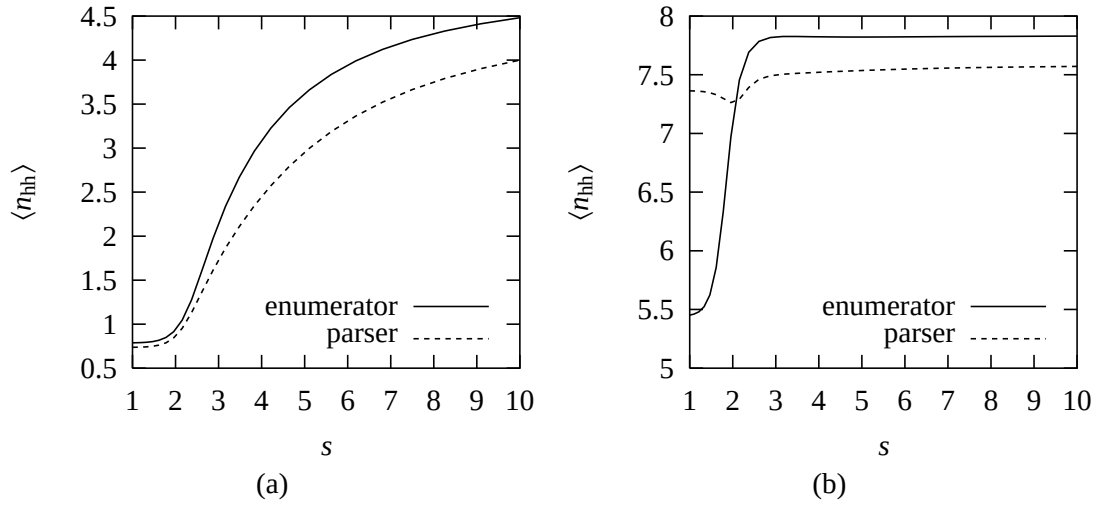


Figure 6.8: Comparison against exact enumeration. Sequence: hpphhpphhpphhpphhpph; hh contacts versus s. (a) $q_{\text{hh}} = 1$. (b) $q_{\text{hh}} = 10$.

enumerator. Figures 6.7 and 6.8 compare the average number of hh contacts; again there is qualitative agreement, except for the region $s < 2$ of Figure 6.8b, which is not very important because such low values for s make helix units unfavorable relative to random coil, which is unrealistic.

We next tried the sequence hpppphhpppphhpppphhppph, which has structures with many hh contacts and structures with many helix units, but not both at the same time (Figure 6.4b). Figures 6.9 and 6.10 compare the average number of helix units; as with the first sequence, the parser's output qualitatively agrees with the exact enumerator's.

Figures 6.11 and 6.12 compare the average number of hh contacts. The agreement is not as good as before for high q_{hh} due to both overcounting of some structures and undercounting of others. For example, the grammar is able to generate small spirals, but does not have the "memory" needed to keep the spiral from colliding with itself. Figure 6.13a shows an unviable conformation generated by the grammar. Such structures are causing the parser to overestimate the average number of hh contacts for high q_{hh} and low s (again, such low values are unrealistic). On the other hand, the grammar does not have a rule that would let helices contact each other at right angles. Figure 6.13a shows a viable conformation with three helices that the grammar does not generate.

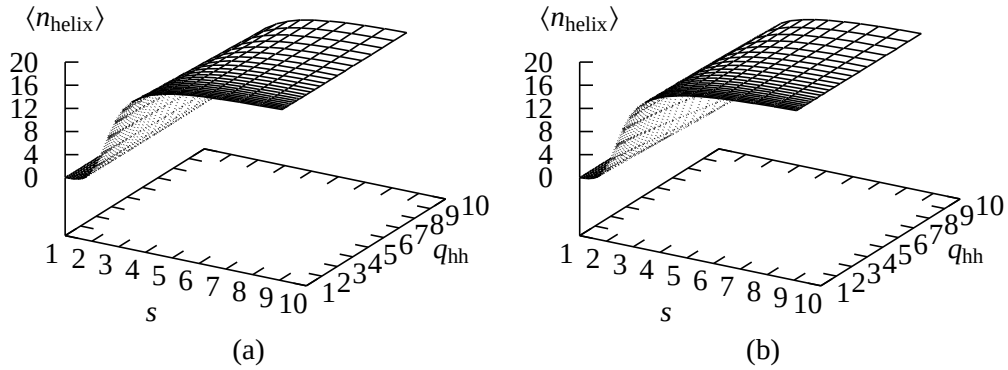


Figure 6.9: Comparison against exact enumeration. Sequence: hpppphhpppphhpppphhppph; helix units versus s and q_{hh}. (a) Exact enumeration. (b) Parser.

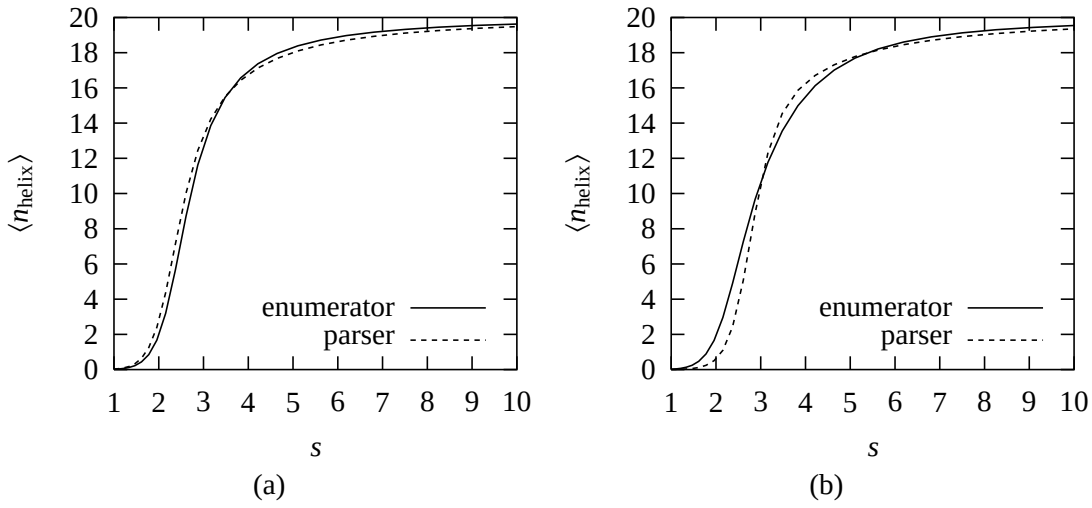


Figure 6.10: Comparison against exact enumeration. Sequence: hpppphhpppphhpppphhppph; helix units versus s. (a) $q_{\text{hh}} = 1$. (b) $q_{\text{hh}} = 10$.

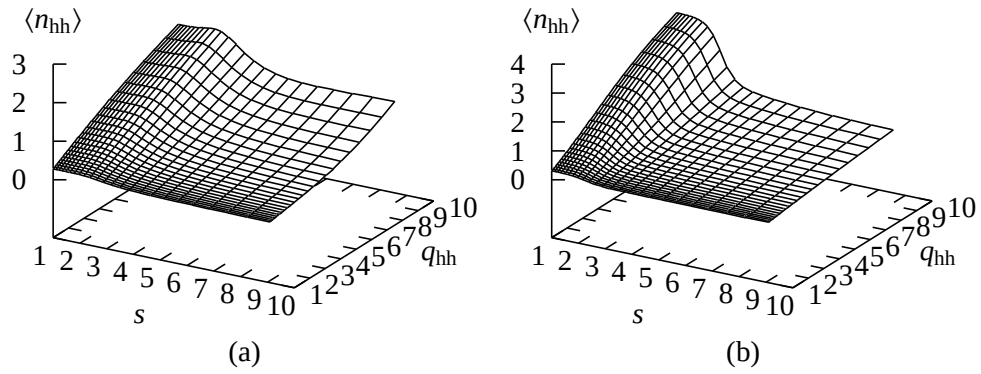


Figure 6.11: Comparison against exact enumeration. Sequence: hpppphhpppphhpppphhppph; hh contacts versus s and q_{hh}. (a) Exact enumeration. (b) Parser.

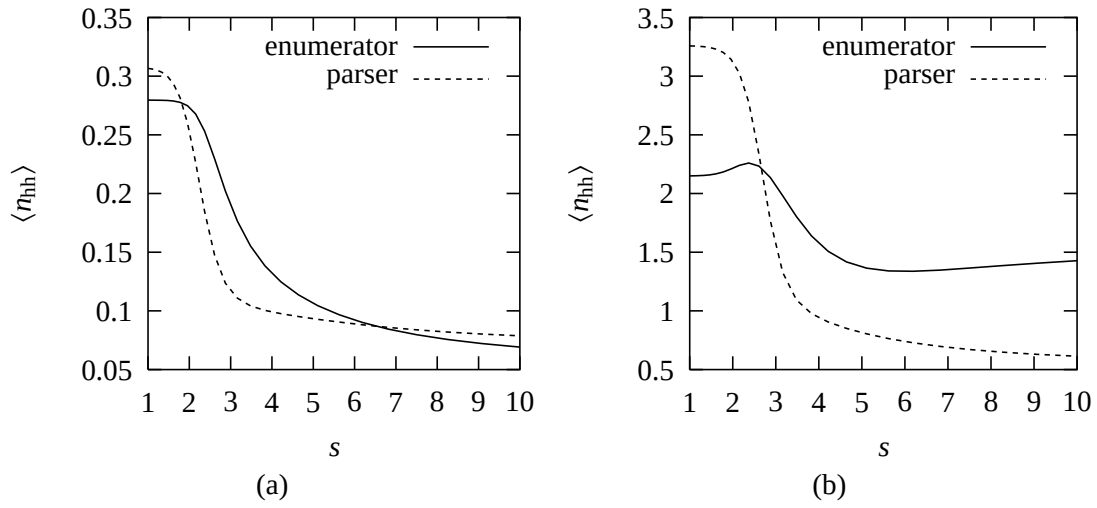
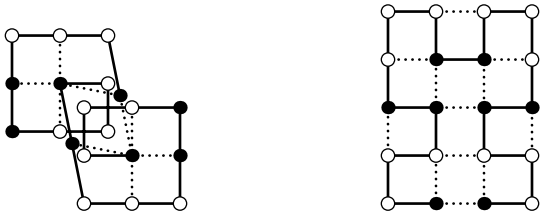


Figure 6.12: Comparison against exact enumeration. Sequence: hpppphhpppphhpppphhppph; hh contacts versus s. (a) $q_{hh} = 1$. (b) $q_{hh} = 10$.



(a) (b)

Figure 6.13: Examples of grammar overcounting (a) and undercounting (b). Hydrophobic monomers (h) are indicated by black circles; polar monomers (p) by white circles.

Missing this structure is causing the parser to underestimate the average number of hh contacts for high q_{hh} and s. Nevertheless, the parser agrees with the enumerator in predicting that the number of hh contacts should decrease with s, in contrast to the first sequence.

The grammar could be modified to try to improve agreement, and this deserves further work. It is possible that in a three-dimensional lattice, the problem of collisions will be less severe because a greater proportion of structures will have viable conformations.

6.3 Intersection in nonlinear sLMGs

In a simple LMG there are no restrictions on what literals may be conjoined in the right-hand side of a production. This makes sLMG closed under intersection: if S_1 and S_2 are the start symbols of two sLMGs G_1 and G_2(with disjoint nonterminal alphabets), create a new start symbol S and add the production

$$S(x) \text{ :- } S_1(x), S_2(x)$$

which recognizes $L(G_1) \cap L(G_2)$. Thus sLMG internalizes the intersection operation, which allows more control than Brown and Wilson's scheme. The caveats from our critique of that scheme still apply, however. For example, Boullier (1999) gives a range concatenation grammar (which has an equivalent sLMG) which he claims models German scrambling, a construction in which all the nouns of a sentence can appear in any order. His grammar checks for a verb for every noun

and vice versa, using intersection to enforce all these constraints simultaneously. But like Brown and Wilson's system, it relies on false assumptions about the generated string to ensure that the constraints are properly coordinated.

Nevertheless, nonlinear sLMG's closure under intersection might be a useful property for modeling complex folds like protein β-sheets. We start with some building blocks:

$$\begin{aligned}
\text{Anti}(\overset{\cdots}{a_1}X, Y\overset{\cdots}{a_2}) &:\!- \text{Anti}(X, Y) && a_i \in \Sigma \\
\text{Anti}(\epsilon, \epsilon) & \\
\text{Par}(\overset{\cdots}{a_1}X, \overset{\cdots}{a_2}Y) &:\!- \text{Par}(X, Y) && a_i \in \Sigma \\
\text{Par}(\epsilon, \epsilon) & \\
\text{Adj}(X, Y) &:\!- \text{Ant}(X, Y) \\
\text{Adj}(X, Y) &:\!- \text{Par}(X, Y)
\end{aligned}$$

The predicates Anti and Par generate pairs of adjacent antiparallel and parallel strands, respectively, and the predicate Adj generates two adjacent strands in either configuration. Irregularities as in Figure 5.16a are also possible, but not shown here.

We can then use the intersection ability of sLMG to combine these pairs of strands into a sheet. Thus the following grammar generates β-sheets where the strands are arranged according to their order in the sequence:

$$\begin{aligned}
\text{Beta}(AB) &:\!- \text{B}(A, B) \\
\text{B}(ABY, B') &:\!- \text{B}(A, B), \text{Adj}(B, B') \\
\text{B}(BY, B') &:\!- \text{Adj}(B, B')
\end{aligned}$$

The first argument to B is a β-sheet minus the last strand, and the second argument is the last strand. The second production forms a larger β-sheet out of a smaller one by appending a new last strand

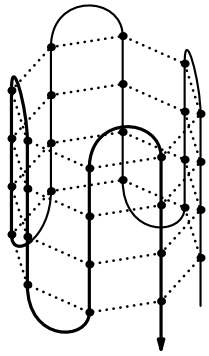


Figure 6.14: β-barrel.

and joining it to the previous last strand using Adj. This production has $O(n^5)$ possible instantiations (because it takes six indices to specify the variables on the left-hand side, but the arguments of B are always adjacent, eliminating one index), and therefore the parsing complexity of this grammar is also $O(n^5)$. Crucially, this complexity bound is not dependent on the number of strands, because each series of contacts is generated in sister subderivations, unlike the multicomponent TAG analysis.

But even sister subderivations can control each other via their root nonterminal (predicate) symbols, as illustrated in the following example. A β-sheet can be rolled into a cylinder to form a *β-barrel* (Figure 6.14). We can generate these as well, but we must keep track of the direction of each strand so as not to generate any Möbius strips, as in the grammar of Figure 6.15. Here B has three arguments: the first strand, the middle part, and the last strand; there is an additional predicate symbol B′ which is the same as B, except that B′ is for sheets with antiparallel first and last strands, whereas B is restricted here to sheets with parallel first and last strands. The first production joins the first and last strands to form a barrel; it uses the information in the B vs. B′ distinction to join the strands so that no Möbius strips will be generated.

The strands of β-sheets do not always appear in linear order; they can be permuted as in Figure 6.16. We can model such permutations by increasing the degree of synchronous parallelism (that is, the number of arguments to B), and therefore increasing parsing complexity. By contrast,

$$\begin{aligned}
\text{Barrel}(ABC) &:\!- \text{B}(A, B, C), \text{Par}(A, C)\\
\text{Barrel}(ABC) &:\!- \text{B}'(A, B, C), \text{Anti}(A, C)\\
\text{B}(A, BCY, C') &:\!- \text{B}'(A, B, C), \text{Anti}(C, C')\\
\text{B}(A, BCY, C') &:\!- \text{B}(A, B, C), \text{Par}(C, C')\\
\text{B}(A, Y, A') &:\!- \text{Par}(A, A')\\
\text{B}'(A, BCY, C') &:\!- \text{B}(A, B, C), \text{Anti}(C, C')\\
\text{B}'(A, BCY, C') &:\!- \text{B}'(A, B, C), \text{Par}(C, C')\\
\text{B}'(A, Y, A') &:\!- \text{Anti}(A, A')
\end{aligned}$$

Figure 6.15: sLMG for β-barrels.

since multicomponent TAG already uses synchronous parallelism to generate all the strands together, it allows permutations of strands at no extra cost.

Suppose we envision a sheet being built up one strand at a time, each successive strand being added to either side of the sheet:

$$\begin{aligned}
\text{Beta}(ABCD) &:\!- \text{B}(A, B, C, D)\\
\text{B}(ABC, D, Y, B') &:\!- \text{B}(A, B, C, D), \text{Adj}(B, B')\\
\text{B}(A, B, CDY, B') &:\!- \text{B}(A, B, C, D), \text{Adj}(D, B')\\
\text{B}(\epsilon, B, Y, B') &:\!- \text{Adj}(B, B')
\end{aligned}$$

Figure 6.16a shows an example sheet that can be generated by this grammar but not the previous ones. In this grammar, the second and fourth arguments to B are the leftmost and rightmost strands (*not* respectively) in the folded structure. The second production adds a new strand on one side, and the third production adds a new strand on the other. Both productions have $O(n^7)$ possible instantiations if we take into account that the four arguments to B will always be adjacent.

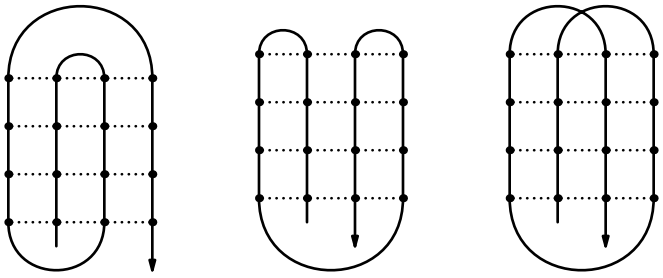


Figure 6.16: Permuted β-sheets.

Suppose we always build up a sheet out of two smaller sheets:

$$\text{Beta}(ABCDE) \text{ :– } \text{B}(A, B, C, D, E)$$

$$\text{B}(ABC, D, EYA', B', C'D'E') \text{ :– } \text{B}(A, B, C, D, E), \text{B}(A', B', C', D', E'), \text{Adj}(B, D')$$

$$\text{B}(A, B, CDEYA', B', C', D', E') \text{ :– } \text{B}(A, B, C, D, E), \text{B}(A', B', C', D', E'), \text{Adj}(D, D')$$

$$\text{B}(\epsilon, B, C, D, \epsilon) \text{ :– } \text{Adj}(B, D)$$

Figure 6.16b shows an example sheet that can be generated by this grammar but not the previous ones. In this grammar, the second and fourth arguments are again the leftmost and rightmost strands (*not* respectively) in the folded structure. The second and third productions join two β-sheets together in two different ways; there are conceivably four ways to join them together, but using only these two avoids spurious ambiguity. Both productions have $O(n^{12})$ possible instantiations if we take into account that the five arguments to B will always be adjacent.

Figure 6.16c shows the only permutation of four strands that the above grammar cannot generate. This does not seem problematic, since, at least for sheets formed out of two hairpin motifs, this permutation was not known as of 1991 to occur in nature (Branden and Tooze, 1999, p. 31).

It should be emphasized, however, that any energies or conformation counts added to these grammars will not be able to make the self-contacts between two strands dependent on self-contacts

with other strands. Akutsu (2000) and Lyngsø and Pedersen (2000) have shown that certain formulations of the problem of predicting RNA secondary structures with generalized pseudoknots are NP-hard. It turns out that both of these proofs assume some kind of dependence between nonadjacent strands. Akutsu assumes that no base can participate in two pairs (one on either side), which is true of RNA secondary structures but not of protein structures. Lyngsø and Pedersen assume that the energy of a base pairing (i, j) can be affected by another base pairing $(j - 1, i')$ even if i and i' are in different strands (or by $(j', i + 1)$ even if j and j' are in different strands); it remains to be seen whether such dependencies might be needed, for example, in calculating conformation counts for β-sheets.

* * *

Intersection is a technique which has been somewhat neglected in computational linguistics, but we have shown in this chapter that it has the potential to provide extra SGC in a way that is useful for analyzing biological sequences. There is a danger, however, of thinking that the extra WGC gained by intersection corresponds with extra SGC. We have explored several ways of employing this technique: we demonstrated a flaw in Brown and Wilson's use of intersections of CFLs for RNA pseudoknots, and we proposed intersections of CFGs and finite-state automata for two-helix bundles and nonlinear sLMGs for larger helix bundles and β-sheets. For future work we would like to extend the helix-bundle work to a three-dimensional lattice and bundles of three or more helices, using a TAG or a nonlinear sLMG. Since these larger bundles can be synthesized in the laboratory (Ken Dill, p.c.), such a model could be evaluated more directly.

Chapter 7

Conclusion

We began this study with the question: What makes one grammar formalism better than another? We developed a theoretical framework for dealing with this question, drawing on ideas from Miller and Joshi and others: in this framework we measure the generative power of grammars by choosing an interpretation domain suited to the task and restricting the interpretation functions to be defined on local domains. We applied this framework in three general areas: statistical parsing, machine translation, and biological sequence analysis.

Statistical parsing The ability of a grammar formalism to describe statistical parsing models is characterized by its SGC with respect to weighted trees (or other weighted structures). This interpretation domain classifies formalisms rather coarsely: of the weakly context-free formalisms we examined, only tree-insertion grammar had greater statistical-modeling power than CFG. But this negative result led to a reinterpretation of lexicalized PCFGs as being cover grammars for grammars resembling TAG. This reinterpretation gave some new insights into how lexicalized PCFG parsers work and how to train them.

We demonstrated a model based on probabilistic tree-insertion grammar with sister-adjunction, discussing implementation details and some of its conceptual advantages. We described how to train this model both using heuristic reconstruction of structural descriptions and using EM to train directly on incomplete structural descriptions. Results on the Penn (English) Treebank were

comparable to lexicalized PCFG parsers, and results on the Penn Chinese Treebank were state-of-the-art.

We also laid a foundation for maximum-entropy models defined on formal grammars, providing an efficient way of estimating maximum-entropy parsers that is more self-contained than parse-reranking models. Investigation of maximum-entropy models based in this way CCG, HPSG, CFG, and TAG is just beginning; further work and comparison with the body of work on stochastic unification-based grammars are needed.

Machine translation The ability of a grammar formalism to describe translations between languages is characterized by its SGC with respect to pairs of strings. By contrast with the previous case, this interpretation domain classifies formalisms rather finely. We argued that this means that the shift in statistical machine-translation research towards syntax-based methods, following a similar shift in statistical parsing research, must be undertaken with greater care. Richer synchronous grammar formalisms may provide the power that statistical machine-translation systems need, but because there are many different levels of power, the right formalism must be chosen. It would be incorrect to conclude from the failure of one formalism that still more powerful formalisms are not worth trying.

We focused one synchronous formalism, synchronous regular-form TAG, as a basis for translation systems. We formally demonstrated that it has greater translation power than other synchronous formalisms recently proposed for statistical machine translation, illustrated its use on some examples, and outlined how it might be integrated into an existing system.

Biological sequence analysis The ability of a grammar formalism to describe secondary/tertiary structures of chain molecules is characterized by its SGC with respect to linked strings. This interpretation domain, like the previous one, classifies formalisms rather finely. We presented a synthesis of previous research in this area, discussing CFG, formalisms coverable by CFG, and formalisms beyond CFG. We then turned our attention to the mechanism of intersection, which has a

more natural interpretation in this domain than with syntactic structures. We showed how to extend our CFG implementation of Chen and Dill's statistical-mechanical model to use a CFG intersected with a finite-state automaton, allowing it to model helix bundles, which was not previously feasible. Finally, we discussed how nonlinear sLMGs might use intersection to model β-sheets more efficiently than previous approaches.

* * *

Grammars are gaining or regaining attention in various quarters of research in natural language processing and structural biology. But the theory fueling these applications will not be fully effective unless it is able to ask and answer the right questions: Is this grammar formalism more powerful than that grammar formalism for this particular application? We have set up a framework for carrying out such comparisons, and used it to explore three areas of application. We hope that the exploration will continue: empirical evaluation of theoretical results in this thesis, new results about other grammar formalisms, and investigation into further areas of application.

References

Abe, Naoki and Hiroshi Mamitsuka. 1997. Predicting protein secondary structures based on stochastic tree grammars. *Machine Learning*, 29:275–301.

Abeille, Anne. 1988. Parsing French with Tree Adjoining Grammar: some linguistic accounts. In *Proceedings of the Twelfth International Conference on Computational Linguistics (COLING)*, pages 7–12.

Abney, Steven P. 1997. Stochastic attribute-value grammars. *Computational Linguistics*, 23:597–618.

Aho, A. V. and J. D. Ullman. 1969. Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*, 3:37–56.

Aho, A. V. and J. D. Ullman. 1971. Translations on a context free grammar. *Information and Control*, 19:439–475.

Akutsu, Tatuya. 2000. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Applied Mathematics*, 104:45–62.

Alberts, Bruce, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. 2002. *Molecular Biology of the Cell*. Garland Science, New York, 4th edition.

Baker, James K. 1979. Trainable grammars for speech recognition. In *Proceedings of the Spring Conference of the Acoustical Society of America*, pages 547–550.

Bar-Hillel, Yehoshua, M. Perles, and E. Shamir. 1964. On formal properties of simple phrase structure grammars. In Yehoshua Bar-Hillel, editor, *Language and Information: Selected Essays on their Theory and Application*. Addison-Wesley, Reading, MA, pages 116–150.

Becker, Tilman, Aravind Joshi, and Owen Rambow. 1991. Long distance scrambling and tree adjoining grammar. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 21–26.

Berger, Adam L., Stephen A. Della Pietra, and Vincent J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22:39–71.

Bertsch, Eberhard and Mark-Jan Nederhof. 2001. On the complexity of some extensions of RCG parsing. In *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT)*, pages 66–77.

Bikel, Daniel M. 2004a. Intricacies of Collins' parsing model. *Computational Linguistics*. To appear.

Bikel, Daniel M. 2004b. *On the Parameter Space of Generative Lexicalized Statistical Parsing Models*. Ph.D. thesis, University of Pennsylvania.

Bikel, Daniel M. and David Chiang. 2000. Two statistical parsing models applied to the Chinese Treebank. In *Proceedings of the Second Chinese Language Processing Workshop*, pages 1–6.

Bikel, Daniel M., Scott Miller, Richard Schwartz, and Ralph Weischedel. 1997. Nymble: a high-performance learning name-finder. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP)*, pages 194–201.

Black, Ezra, Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer, and Salim Roukos. 1992. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 5th DARPA Speech and Natural Language Workshop*, pages 31–37.

Booth, Taylor and Richard Thompson. 1973. Applying probability meausres to abstract languages. *IEEE Transactions on Computers*, C-22(5):442–450.

Boullier, Pierre. 1999. Chinese numbers, MIX, scrambling, and range concatenation grammars. In *Proceedings of the Ninth Conference of the European chapter of the Association for Computational Linguistics (EACL)*, pages 53–60.

Boullier, Pierre. 2000. Range concatenation grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT)*, pages 53–64.

Branden, Carl-Ivar and John Tooze. 1999. *Introduction to Protein Structure*. Garland Publishing, Inc., New York, 2nd edition.

Bresnan, Joan, Ronald M. Kaplan, P. Stanley Peters, and Annie Zaenen. 1982. Cross-serial dependencies in Dutch. *Linguistic Inquiry*, 13:613–635.

Brown, Michael and Charles Wilson. 1996. RNA pseudoknot modeling using intersections of stochastic context free grammars with applications to database search. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 109–125.

Brown, Peter F., Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19:263–311.

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

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 132–139.

Chen, John and K. Vijay-Shanker. 2000. Automated extraction of TAGs from the Penn Treebank. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT)*, pages 65–76.

Chen, Shi-Jie and Ken A. Dill. 1995. Statistical thermodynamics of double-stranded polymer molecules. *J. Chem. Phys.*, 103(13):5802–5813.

Chen, Shi-Jie and Ken A. Dill. 1998. Theory for the conformational changes of double-stranded chain molecules. *J. Chem. Phys.*, 109(11):4602–4616.

Chen, Shi-Jie and Ken A. Dill. 2000. RNA folding energy landscapes. *Proceedings of the National Academy of Sciences*, 97(2):646–651.

Chiang, David. 2000. Statistical parsing with an automatically-extracted tree adjoining grammar. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 456–463.

Chiang, David. 2001. Constraints on strong generative power. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 124–131.

Chiang, David. 2002. Putting some weakly context-free formalisms in order. In *Proceedings of the Sixth International Workshop on TAG and Related Formalisms (TAG+)*, pages 11–18.

Chiang, David. 2003. Mildly context sensitive grammars for estimating maximum entropy parsing models. In *Proceedings of Formal Grammar 2003*. Final version of proceedings to appear.

Chiang, David and Aravind K. Joshi. 2002. Formal grammars for estimating partition functions of double-stranded chain molecules. In *Proceedings of the Second International Conference on Human Language Technology Research (HLT)*, pages 63–67.

Chiang, David, William Schuler, and Mark Dras. 2000. Some remarks on an extension of synchronous TAG. In *Proceedings of the Fifth International Workshop on TAG and Related Formalisms (TAG+)*, pages 61–66.

Chiou, Fu-Dong, David Chiang, and Martha Palmer. 2001. Facilitating treebank annotation using a statistical parser. In *Proceedings of the First International Conference on Human Language Technology Research (HLT)*, pages 117–120. Poster session.

Chomsky, Noam. 1963. Formal properties of grammars. In R. Duncan Luce, Robert R. Bush, and Eugene Galanter, editors, *Handbook of Mathematical Psychology*, volume 2. Wiley, New York, pages 323–418.

Chomsky, Noam and George A. Miller. 1963. Introduction to the formal analysis of natural languages. In R. Duncan Luce, Robert R. Bush, and Eugene Galanter, editors, *Handbook of Mathematical Psychology*, volume 2. Wiley, New York, pages 269–321.

Clark, Stephen and James R. Curran. 2003. Log-linear models for wide-coverage CCG parsing. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 97–104, Sapporo, Japan.

Collins, Michael. 1997. Three generative lexicalised models for statistical parsing. In *Proceedings of ACL-EACL*, pages 16–23.

Collins, Michael. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Collins, Michael. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pages 175–182. Morgan Kaufmann.

Collins, Michael and James Brooks. 1995. Prepositional phrase attachment through a backed-off model. In *Proceedings of the Third Workshop on Very Large Corpora (WLVC)*, pages 27–38.

Darroch, J. N. and D. Ratcliff. 1972. Generalized iterative scaling for log-linear models. *Annals of Mathematical Statistics*, 43:1470–1480.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38.

Dras, Mark. 1999. A meta-level grammar: redefining synchronous TAG for translation and paraphrase. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 80–87.

Eisner, Jason. 2003. Learning non-isomorphic tree mappings for machine translation. In *Companion Volume to the Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 205–208. Poster/demonstration session.

Geman, Stuart and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 279–286.

Gildea, Daniel. 2001. Corpus variation and parser performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 167–202 (*sic*).

Gildea, Daniel. 2003. Loosely tree-based alignment for machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 80–87.

Goodman, Joshua. 1997. Global thresholding and multiple-pass parsing. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 11–25.

Goodman, Joshua. 1999. Semiring parsing. *Computational Linguistics*, 25:573–605.

Groenink, Annius Victor. 1997. *Surface without Structure: Word order and tractability issues in natural language analysis*. Ph.D. thesis, University of Utrecht.

Hopcroft, John E. and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.

Huybregts, M. A. C. 1976. Overlapping dependencies in Dutch. *Utrecht Working Papers in Linguistics*, 1:24–65.

Huybregts, Riny. 1984. The weak inadequacy of context-free phrase structure grammars. In G. de Haan, M. Trommele, and W. Zonneveld, editors, *Van Periferie naar Kern*. Foris, Dordrecht, pages 81–99.

Hwa, Rebecca. 1998. An empirical evaluation of probabilistic lexicalized tree insertion grammars. In *Proceedings of COLING-ACL*, pages 557–563.

Hwa, Rebecca. 2001. *Learning Probabilistic Lexicalized Grammars for Natural Language Processing*. Ph.D. thesis, Harvard University.

Johnson, Mark. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24:613–632.

Johnson, Mark, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 535–541.

Joshi, A. K., L.S. Levy, and M. Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10:136–163.

Joshi, Aravind K. 1985. Tree adjoining grammars: How much context-sensitivity is necessary for assigning structural descriptions? In David Dowty, Lauri Karttunen, and Arnold Zwicky, editors, *Natural Language Parsing*. Cambridge University Press, Cambridge, pages 206–250.

Joshi, Aravind K. 2003. A note on the strong and weak generative powers of formal systems. *Theoretical Computer Science*, 293:243–259. Originally presented at TAG+5 in 2000.

Joshi, Aravind K. and Yves Schabes. 1997. Tree-adjoining grammars. In Grzegorz Rosenberg and Arto Salomaa, editors, *Handbook of Formal Languages and Automata*, volume 3. Springer-Verlag, Heidelberg, pages 69–124.

Kato, Yuki, Hiroyuki Seki, and Tadao Kasami. 2004. Subclasses of tree adjoining grammar for RNA secondary structure. In *Proceedings of the Seventh International Workshop on TAG and Related Formalisms (TAG+)*, pages 48–55.

Klein, Dan and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 423–430.

Knuth, Donald E. 1968. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145. Corrections published in 5(1):95–96.

Kroch, Anthony S. and Beatrice Santorini. 1991. The derived structure of the West Germanic verb raising construction. In Robert Freidin, editor, *Principles and Parameters in Comparative Grammar*. MIT Press, Cambridge, MA, pages 269–338.

Kuroda, S.-Y. 1976. A topological study of phrase-structure languages. *Information and Control*, 30:307–379.

Kuroda, S.-Y. 1987. A topological approach to structural equivalence of formal languages. In Alexis Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins, pages 173–189.

Lafferty, John, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, pages 282–289. Morgan Kaufmann.

Lambek, Joachim. 1958. The mathematics of sentence structure. *American Mathematical Monthly*, 65(3):154–170.

Lari, K. and S. J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.

Lau, Kit Fun and Ken A. Dill. 1989. A lattice statistical mechanics model of the conformation and sequence spaces of proteins. *Macromolecules*, 22:3986–3997.

Lyngsø, Rune B. and Christian N. S. Pedersen. 2000. RNA pseudoknot prediction in energy-based models. *Journal of Computational Biology*, 7(3/4):409–427.

Madras, Neal and Gordon Slade. 1993. *The Self-Avoiding Walk*. Birkhäuser, Boston.

Magerman, David M. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 276–283.

Malouf, Robert. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Conference on Natural Language Learning (CoNLL)*, pages 49–55.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19:313–330.

Miller, Philip H. 1999. *Strong Generative Capacity: The Semantics of Linguistic Formalism*. CSLI Publications, Stanford.

Miller, Scott, Heidi Fox, Lance Ramshaw, and Ralph Weischedel. 2000. A novel use of statistical parsing to extract information from text. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 226–233.

Miyao, Yusuke, Takashi Ninomiya, and Jun'ichi Tsujii. 2003. Probabilistic modeling of argument structures including non-local dependencies. In *Proceedings of the Conference on Recent Advances in Natural Language Processing (RANLP-2003)*, pages 285–291.

Miyao, Yusuke and Jun'ichi Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proceedings of the Second International Conference on Human Language Technology Research (HLT)*, pages 292–297.

Neumann, Günter. 1998. Automatic extraction of stochastic lexicalized tree grammars from treebanks. In *Proceedings of the Fourth International Workshop on TAG and Related Formalisms (TAG+)*, pages 120–123.

Nijholt, Anton. 1980. *Context-Free Grammars: Covers, Normal Forms, and Parsing*. Springer-Verlag, Berlin.

Pentus, Mati. 1993. Lambek grammars are context-free. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, pages 429–433, Los Alamitos, CA.

Pentus, Mati. 2003. Lambek calculus is NP-complete. Technical Report TR-2003005, CUNY Ph.D. Program in Computer Science.

Post, Emil L. 1943. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65:197–215.

Pullum, Geoffrey K. and Gerald Gazdar. 1982. Natural languages and context-free languages. *Linguistics and Philosophy*, 4:471–504.

Rambow, Owen and Giorgio Satta. 1999. Independent parallelism in finite copying parallel rewriting systems. *Theoretical Computer Science*, 223:87–120.

Rambow, Owen, K. Vijay-Shanker, and David Weir. 1995. D-tree grammars. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 151–158.

Ratnaparkhi, Adwait. 1996. A maximum-entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 133–142.

Ratnaparkhi, Adwait. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–10.

Resnik, Philip. 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of the Fourteenth International Conference on Computational Linguistics (COLING)*, pages 418–424.

Rivas, Elena and Sean R. Eddy. 2000. The language of RNA: a formal grammar that includes pseudoknots. *Bioinformatics*, 16(4):334–340.

Rogers, James. 1994. Capturing CFLs with tree adjoining grammars. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 155–162.

Rogers, James. 2003. Syntactic structures as multi-dimensional trees. *Research on Language and Computation*, 1:265–305.

Sakakibara, Yasubimi, Michael Brown, Richard Hughey, I. Saira Mian, Kimmen Sjölander, Rebecca C. Underwood, and David Haussler. 1994. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research*, 22:5112–5120.

Satta, Giorgio and William Schuler. 1998. Restrictions on tree adjoining languages. In *Proceedings of COLING-ACL*, pages 1176–1182.

Schabes, Yves. 1990. *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, University of Pennsylvania.

Schabes, Yves. 1992. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the Fourteenth International Conference on Computational Linguistics (COLING)*, pages 426–432.

Schabes, Yves, Anne Abeille, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to Tree Adjoining Grammars. In *Proceedings of the Twelfth International Conference on Computational Linguistics (COLING)*, pages 578–583.

Schabes, Yves and Stuart M. Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20:91–124.

Schabes, Yves and Richard C. Waters. 1993. Lexicalized context-free grammars. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 121–129.

Schabes, Yves and Richard C. Waters. 1995. Tree insertion grammar: a cubic-time parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*, 21:479–513.

Schabes, Yves and Richard C. Waters. 1996. Stochastic lexicalized tree-insertion grammar. In Harry Bunt and Masaru Tomita, editors, *Recent Advances in Parsing Technology*. Kluwer, Dordrecht, pages 281–294.

Schuler, William. 1999. Preserving semantic dependencies in synchronous tree adjoining grammar. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 88–95.

Schuler, William, David Chiang, and Mark Dras. 2000. Multi-component TAG and notions of formal power. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 448–455.

Searls, David B. 1992. The linguistics of DNA. *American Scientist*, 80:579–591.

Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–344.

Shieber, Stuart M. 1994. Restricting the weak generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, 10(4):371–385. Special Issue on Tree Adjoining Grammars.

Shieber, Stuart M. and Yves Schabes. 1990. Synchronous tree-adjoining grammars. In *Proceedings of the Thirteenth International Conference on Computational Linguistics (COLING)*, volume 3, pages 1–6.

Shieber, Stuart M., Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36.

Uemura, Yasuo, Aki Hasegawa, Satoshi Kobayashi, and Takashi Yokomori. 1999. Tree adjoining grammars for RNA structure prediction. *Theoretical Computer Science*, 210:277–303.

Vijay-Shanker, K. and David J. Weir. 1993. The use of shared forests in tree adjoining grammar parsing. In *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 384–393.

Weir, David J. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, University of Pennsylvania.

Wu, Dekai. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23:377–404.

Xia, Fei. 1999. Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99)*, pages 398–403.

Xia, Fei, Martha Palmer, Nianwen Xue, Mary Ellen Okurowski, John Kovarik, Fu-Dong Chiou, Shizhe Huang, Tony Kroch, and Mitch Marcus. 2000. Developing guidelines and ensuring consistency for Chinese text annotation. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC-2000)*, pages 3–10, Athens, Greece.

Yamada, Kenji and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 523–530.

Zimm, B. H. and J. K. Bragg. 1959. Theory of the phase transition between helix and random coil in polypeptide chains. *Journal of Chemical Physics*, 31:526–535.

Index