Synchronous and Multicomponent Tree-Adjoining Grammars: Complexity, Algorithms and Linguistic Applications

A dissertation presented

by

Rebecca Nancy Nesson

to

The Department of Computer Science

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in the subject of

Computer Science

Harvard University

Cambridge, Massachusetts

May 2009

Thesis advisor

Stuart M. Shieber

Author

Rebecca Nancy Nesson

Synchronous and Multicomponent Tree-Adjoining Grammars: Complexity, Algorithms and Linguistic Applications

Abstract

This thesis addresses the design of appropriate formalisms and algorithms to be used for natural language processing. This entails a delicate balance between the ability of a formalism to capture the linguistic generalizations required by natural language processing applications and the ability of a natural language processing application based on the formalism to process the formalism efficiently enough to be useful. I focus on the Tree-Adjoining Grammar formalism as a base and on the mechanism of grammar synchronization for managing relationships between the input and output of a natural language processing system.

Grammar synchronization is a formal concept by which the derivations of two distinct grammars occur in tandem so that a single isomorphic derivation produces distinct derived structures in each of the synchronized grammars. Using synchronization implies a strong assumption—one that I seek to justify in the second part of the thesis—namely that certain critical relationships in natural language applications, such as the relationship between the syntax and semantics of a language or the relationship between the syntax of two natural languages, are close enough to be expressed with grammars that share a derivational structure.

The extent of the isomorphism between the derived structures of the related lan-

guages is determined only in part by the synchronization. The base formalism chosen can offer greater or lesser opportunity for divergence in the derived structures. My choice of a base formalism is motivated directly by research into applications of synchronous TAG-based grammars to two natural language applications: semantic interpretation and natural language translations. I first examine a range of TAG variants that have not previously been studied in this level of detail to determine their computational properties and to develop algorithms that can be used to process them. Original results on the complexity of these formalisms are presented as well as novel algorithms for factorizing grammars to reduce the time required to process them. In Part II, I develop applications of synchronous Limited Delay Tree-Local Multicomponent TAG to semantic interpretation and probabilistic synchronous Tree Insertion Grammar to statistical natural language translation.

Contents

Citations to Previously Published Work

Chapter 2 is based in large part on an article currently in submission. A version is currently available as a technical report [Nesson et al., 2008b]. Large portions of Chapter 3 are based on Nesson and Shieber [2009]. Chapter 4 is based on Nesson et al. [2008a]. Various portions of Chapters 5 and 6 draw on material from several papers [Nesson and Shieber, 2008, 2007, 2006]. Some portions of Chapter 7 are based on Nesson et al. [2005].

Acknowledgments

This dissertation is the culmination of six years graduate education in computer science and thirty years of formal education in total. Needless to say, many people deserve my gratitude.

First and foremost, I would like to thank my advisor, Stuart Shieber. He was the professor of my first course in computer science, which I took in my third year of law school. A few months into that semester I went to his office hours to tell him that CS50 had convinced me that I wanted to enroll in a Ph.D. in computer science and to ask his advice. Although he was soberingly skeptical at that first meeting, he welcomed me in his computational linguistics course the next year and the following year, in spite of my limited computer science background, he accepted my application to work with him on a Ph.D. Throughout graduate school he has been an ideal advisor. He helped me to find and define interesting research questions and has always engaged both the high level and the details of our work. He also respected the diversity of my interests, even when they were time-consuming, and encouraged and supported me through getting married and having two children. I have learned an enormous amount from him and had a good time doing it. He is the originator of the big ideas underlying this thesis and was a true collaborator on every piece of this work.

I would also like to thank all of the professors and teachers and fellow students who have supported me in graduate school. Professors Barbara Grosz, Harry Lewis, and Henry Leitner in particular have believed in me, given me their time, advising and advice, and repeatedly employed me as a teaching fellow in their courses. My dissertation committee, Barbara Grosz, Michael Collins, and Stuart Shieber encour-

aged me in this work and contributed helpful comments and advice that improved the final product.

Almost every chapter in this thesis is the result of collaboration with other scholars. I would like to thank Giorgio Satta who worked with me on many of the results presented in the first part of the dissertation. I admired his work for years before I got a chance to work with him myself. He is the acute thinker about computational complexity that I expected and is also kind, funny, and easy to work with even over great distance. Alexander Rush, an undergraduate (at that time) whose abilities both in research and engineering far exceeded that of the typical undergraduate (and graduate) student, collaborated with us on the work on translation that appears in Chapter 7. Although the resulting work does not appear in this dissertation, I would also like to thank Floris Roelofsen and Barbara Grosz for collaborating with me on work on centering theory and anaphora resolution. That work was among the most exciting I did in graduate school and the underlying thinking is responsible for some of the ideas about implementing binding theory that appear in Chapter 5. Lastly, I'd like to thank the research groups of Aravind Joshi, Maribel Romero, and Laura Kallmeyer for an encouraging and motivating dialogue about Tree-Adjoining Grammar and its application to natural language semantics.

The Berkman Center for Internet & Society has been my unofficial home at Harvard since I began at Harvard Law School more than ten years ago. Jonathan Zittrain showed me the excitement of studying the Internet and technology and has been an invaluable advisor and friend ever since. From the time we were fellow students in Jonathan's class, John Palfrey has been a great friend, co-teacher, and supporter of

my work and projects. My father, Charles Nesson, has been my biggest advocate in general and my collaborator on education and technology projects. The whole Berkman team has been a wonderful partner in my extracurricular projects over the past six years.

Finally, I'd like to thank my family. In spite of my inability to truly explain to them what I was working on, they have trusted me that the work was worthwhile and supported me every step of the way with love, food, babysitting, and everything else. I would particularly like to thank my parents, Fern and Charlie Nesson, for giving me a love of learning and a great education. I have loved being a student from the beginning and it is not without regret that I leave that phase of my life behind.

For my daughters Nico and Charlie

who encouraged me to finish with many small kicks to the belly

Chapter 1

Introduction

Many natural language processing tasks can be characterized as determining the relationship between two languages. For instance, the natural language translation task relates the syntax of two languages. Semantic computation relates the strings of a language or the syntax of a language to its semantics. In this thesis I explore the formal mechanism of grammar synchronization as a method for capturing a complex but close relationship between two languages. Grammar synchronization offers a way to separate the language-specific differences between two related languages from the interface between them. In a synchronous grammar the derivations of two distinct grammars occur in tandem so that a single isomorphic derivation produces distinct derived structures in each of the synchronized grammars. Using synchronization implies a strong assumption–one that I seek to justify in the second part of the thesis–namely that certain critical relationships in natural language applications, such as the relationship between the syntax and semantics of a language or the relationship between the syntax of two natural languages, are close enough to be expressed with

grammars that share a derivational structure but are still sufficiently constrained in complexity that they can be processed reasonably efficiently.

The extent of the isomorphism between the derived structures of the related languages is determined only in part by the synchronization. The base formalism chosen can offer greater or lesser opportunity for divergence in the derived structures. I focus on the Tree-Adjoining Grammar (TAG) formalism as the underlying basis for synchronization for several reasons. First, and by design, its basic operations of adjunction and substitution naturally model the primary linguistic operations of modification and argument substitution. Second, it is widely used and studied in applications to natural language syntax and has been proposed for use in semantics and translation but has not prior to the last several years been thoroughly examined for applications in these areas. Third, it falls in the category of mildly context-sensitive grammar formalisms which, as a starting point, strike a good balance between expressivity and computational tractability.

The appropriateness of the choice of TAG as a base formalism is borne out both by the complexity analyses and processing algorithms for TAG variants presented in Part I and by the successful application of synchronous TAG-based grammars to two natural language processing tasks—semantic interpretation and natural language translation—in Part II. In Part I, I carefully examine a range of TAG variants that have not previously been studied in this level of detail to determine their computational properties and to develop algorithms that can be used to process them. In Part II, I develop applications of synchronous Limited Delay Tree-Local Multicomponent TAG to semantic interpretation and probabilistic synchronous Tree Insertion

Grammar to statistical natural language translation.

The remainder of this chapter provides a technical introduction to TAG and its multicomponent variants that motivates their use in computational linguistics and sets the stage for the work presented in the following chapters. It is structured as follows. Section 1.1 introduces the Tree-Adjoining Grammar formalism. In Section 1.2 I introduce the inference rule-based method for parser description used throughout the thesis using a CKY-style TAG parser. In Section 1.3 I introduce multicomponent TAGs. Synchronization and synchronous TAG are introduced in Section 1.4. In Section 1.5 I describe the contents and main contributions of the first part of the thesis. In Section 1.6 I introduce the applications that will be discussed in the second part of the thesis and discuss the contributions to current research made by that part.

1.1 Tree Adjoining Grammar

Tree adjoining grammar (TAG) was introduced by Joshi [1985] for use in computational linguistics because of its ability to naturally characterize the linguistic operations of argument substitution and optional modification without sacrificing computational efficiency. TAGs consist of elementary structures that, in syntactic applications, resemble fragments of parse trees and generally represent members of the lexicon of a natural language. These elementary structures are combined using two operations: **substitution**, which models argument substitution, and **adjunction**, which models optional modification and predication relations. The inclusion of the adjunction operation makes TAG more expressive than context-free grammar (CFG) and allows it to naturally model certain linguistic phenomena such as long

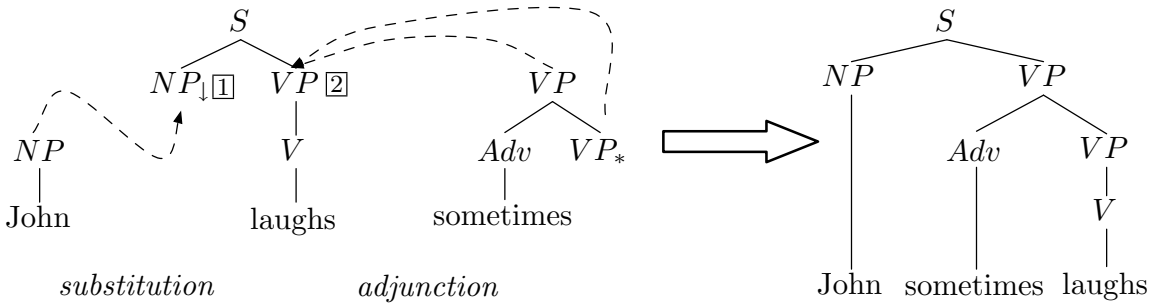


Figure 1.1: An example of TAG operations **substitution** and **adjunction** used here to model natural language syntax.

distance dependencies, that are difficult to model elegantly using CFG. The natural modeling of syntactic relationships and the relatively tractable computational properties of TAG have made it a popular formalism choice for computational linguistics working on natural language syntax since its introduction.

A TAG consists of a set of elementary tree structures of arbitrary depth, which are combined with two operations, substitution and adjunction. Internal nodes in the elementary trees are labeled with a nonterminal symbol. Frontier nodes may be labeled with either terminal symbols or nonterminal symbols annotated with one of the diacritics $\downarrow$ or $*$. The $\downarrow$ diacritic marks a frontier nonterminal node as a **substitution node**, the target of the substitution operation. The **substitution** operation occurs when an elementary tree rooted in a nonterminal symbol A replaces a substitution node with the same nonterminal symbol.

Auxiliary trees are elementary trees in which the root and a frontier node, called the **foot node** and distinguished by the diacritic $*$, are labeled with the same nonterminal A. The path from root to foot is called the **spine**. The **adjunction** operation involves splicing an auxiliary tree in at an internal node in an elementary

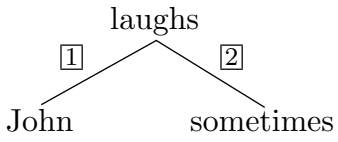


Figure 1.2: The derivation tree for the example shown in Figure 1.1. Note that because we use explicitly notated links, the branches of the tree are labeled by the links at which operations occur rather than by the addresses at which they occur.

tree also labeled with nonterminal A. Trees without a foot node, intended for substitution rather than adjunction into other trees, are called **initial trees**. Examples of the substitution and adjunction operations on sample elementary trees are shown in Figure 1.1.

A TAG derivation can be fully specified by a **derivation tree**, which records how the elementary structures are combined using the TAG operations to form the derived tree. The nodes of the derivation tree are labeled by the names of the elementary trees and the edges are labeled by the addresses at which the child trees substitute or adjoin. In contrast to CFG, the derivation and derived trees of a TAG may be distinct. The derivation tree for the example shown in Figure 1.1 is given in Figure 1.2.

We depart from the traditional definition in notation only by specifying adjunction sites explicitly with numbered links.[1] Each link may be used only once in a derivation. A numbered link at a single site in a tree specifies that a single adjunction is available at that site. Only a single link is permitted at a given node. Because we use explicit links, the edges in the derivation tree are labeled with the number of the link used rather than the address at which the operation takes place.

[1]This is done in order to simplify the presentation of the issues raised by multicomponent adjunctions, which will be introduced in Section 1.3.

Multiple adjunction refers to permitting an unbounded number of adjunctions to occur at a single adjunction site [Vijay-Shanker, 1987, Shieber and Schabes, 1994]. Although multiple adjunction appears to be an obvious way to handle certain cases of modification, in the standard definition of TAG, multiple adjunction is disallowed in order to avoid spurious ambiguity. Instead, where several auxiliary trees might attach at the same location, the usual TAG method is to adjoin one auxiliary tree to another to form a chain. However, as demonstrated by examples of adverbial and adjectival modification and quantifier scope ambiguities, Shieber and Schabes [1994] convincingly argue that the use of multiple adjunction is warranted when the adjoining auxiliary tree models optional modification (as opposed to serving as a predicative modifier). Both their argument and the contrast between the standard and multiple adjunction methods are demonstrated well by the following examples from their paper:

(1) a. Brockway walked his Labrador yesterday.

b. Brockway walked his Labrador yesterday towards the yacht club.

(2) a. Brockway resembled his Labrador yesterday.

b. * Brockway resembled his Labrador yesterday towards the yacht club.

Figure 1.3 contrasts the standard TAG chain method of modification with a derivation that uses multiple adjunction. The direct relationship between the adverbial phrase *towards the yacht club* and the verb that could be used to elegantly draw the distinction between examples (2) and (1) is present only in the derivation that uses multiple adjunction. In Chapter 5 we build substantially on the suggestion of using

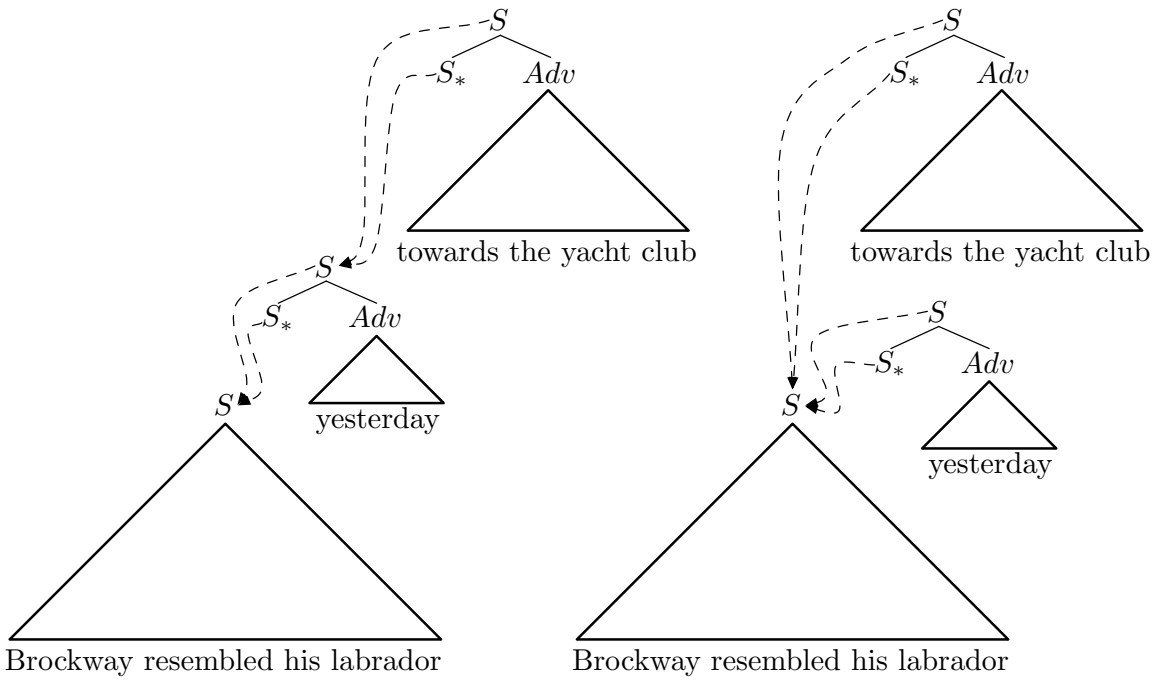


Figure 1.3: Example contrasting the standard TAG chaining method of modification (left) with the multiple adjunction method of modification (right). This example is drawn from Shieber and Schabes [1994] and illustrates that it may be desirable to permit multiple adjunction in order to directly capture constraints on modification between the modifier and the modified.

multiple adjunction to model scope ambiguity in natural language semantics.

1.1.1 Feature-Based TAG

A commonly used extension to TAG is the addition of feature structures on nodes to manage certain grammatical relationships, such as agreement or case checking. As long as the set of feature values is finite, they are computationally benign. Features are implemented in TAG with a top and bottom feature on each node. When an

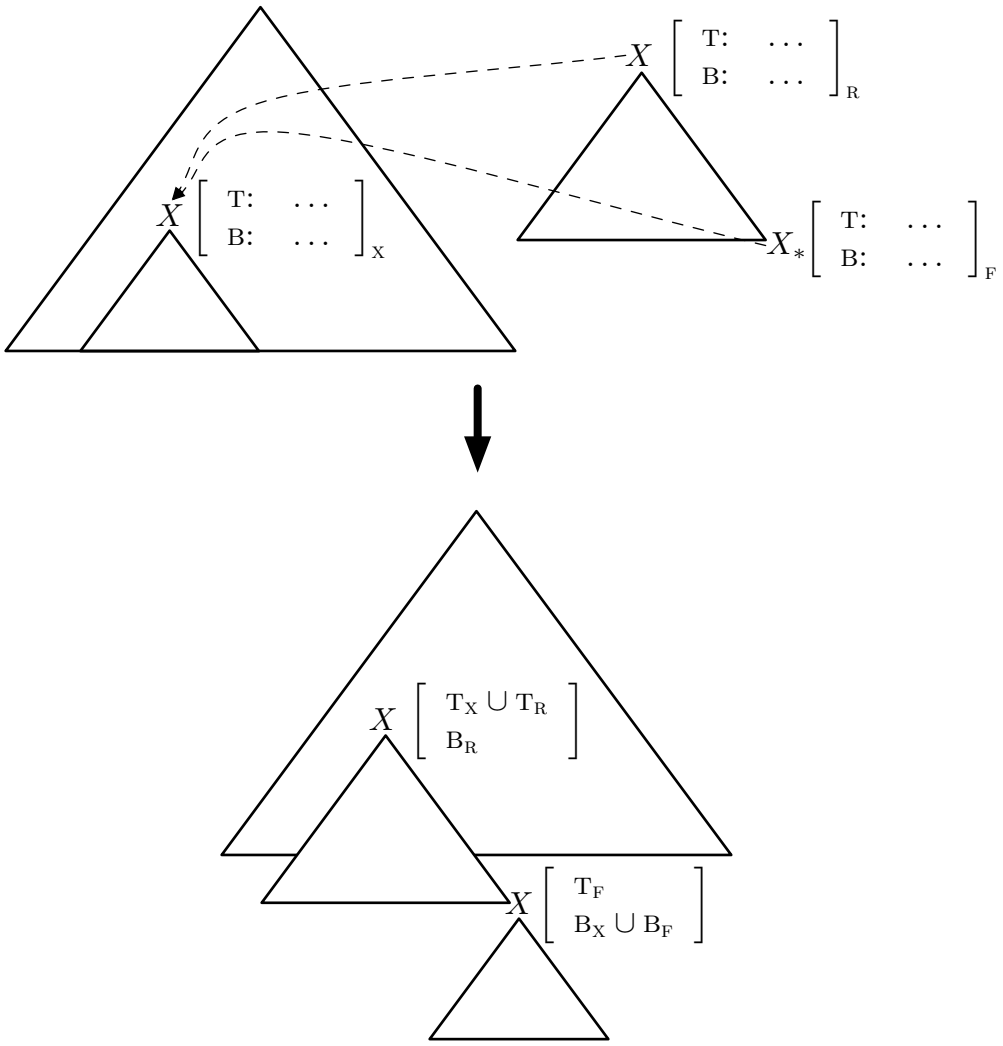


Figure 1.4: An example of feature unification in feature-based TAG. Each node has a top and bottom feature structure. When an adjunction occurs, the top (resp. bottom) feature of the root (resp. foot) of the adjoining tree unifies with the top (resp. bottom) feature of the node where it adjoins. When all operations are complete, the top and bottom features of each node unify.

auxiliary tree adjoins to another tree the top feature of its root node unifies with the top feature of the adjunction site. The bottom feature of its foot node unifies with the bottom feature of the adjunction site. Once all operations are complete, top-bottom unification is performed so that the top and bottom features of each node in the

derived tree are unified. If any step in this unification process produces a mismatch between feature values, the unification fails. Feature unification is demonstrated in Figure 1.4. For readability we generally omit the feature structures in the lexical entries used in our analyses in Part II of this thesis, it is assumed that features are present and are used for the standard grammatical tasks. When we use them explicitly in an analysis, their content is included.

1.2 TAG Parsing

In this thesis we explore the applicability of TAG variants to computation of natural language semantics in conjunction with syntactic analysis and natural language translation. The fundamental computational task required to accomplish these tasks is the assignment of structure to the input sentences: parsing. As briefly noted above, one of the appealing features of TAG is that it may be parsed reasonably efficiently. In this section we introduce one notation for specifying parsers (inference-rule-based parsing) and one algorithm for parsing TAG (CKY) and use it as an opportunity to explicate the notation and methods used for all parsing algorithms that appear in this thesis.

The Cocke-Kasami-Younger (CKY) algorithm is a simple bottom-up parsing algorithm. The simplicity of the algorithm makes it a desirable starting point for designing and explaining new parsing algorithms. It was first introduced for use with context-free grammars in Chomsky normal form [Kasami, 1965, Younger, 1967]. Shieber et al. [1995] and Vijay-Shanker [1987] provide CKY-style TAG parsing algorithms.

Shieber et al. [1995] introduce an inference-rule-based notation for specifying

Item Form:	$\langle A, i, j \rangle$	
Goal Item:	$\langle S, 0, n \rangle$	$\mathsf{len} = n$
Axiom:	$\langle A, i, i+1 \rangle$	$A \rightarrow w_{i+1}$
Inference Rule:	$\dfrac{\langle B, i, j \rangle \langle C, j, k \rangle}{\langle A, i, k \rangle}$	$A \rightarrow B\ C$

Figure 1.5: The CKY parsing algorithm for Chomsky normal form CFG.

parsers in which a parser may be fully defined by a set of axioms and inference rules with consequents that serve as the items in a standard chart-parsing framework. Using their notation, CKY-style CFG parsing can be implemented in a chart-parsing framework with just one axiom and one inference rule (for grammars in Chomsky normal form) as shown in Figure 1.5. Items consist of a nonterminal symbol and two indices that indicate the span of the target sentence that can be parsed with that nonterminal at the root of the parse tree. The axiom states that an item should be added for each word in the target sentence that contains the nonterminal of the rule in the grammar that produces that word and the indices of the location of the word in the target sentence. The inference rule states that if two items, rooted in nonterminals B and C have adjacent spans and there exists a rule $A \rightarrow B\ C$, then a new item is produced with nonterminal A and the combined span of the two antecedent items. Figure 1.6 demonstrates an example of the application of the parsing algorithm to the sentence *John sometimes reads the paper* using the grammar given.

Shieber et al. [1995] and Vijay-Shanker [1987] generalize the CKY algorithm for application to TAG. We present the algorithm from Shieber et al. [1995] with minor modification here. As shown in Figure 1.7, items in CKY-style TAG parsing consist

$_0$ John $_1$ sometimes $_2$ reads $_3$ the $_4$ paper $_5$

S	$\rightarrow$	$NP\ VP$	NP	$\rightarrow$	John
VP	$\rightarrow$	$V\ NP$	Adv	$\rightarrow$	sometimes
VP	$\rightarrow$	$Adv\ VP$	V	$\rightarrow$	reads
NP	$\rightarrow$	$Det\ N$	Det	$\rightarrow$	the
			N	$\rightarrow$	paper

	Rule	Rule Application	Side Condition	Sentence Fragment
1.	Axiom	$\langle NP, 0, 1\rangle$	NP $\rightarrow$ John	$_0$ John $_1$
2.	Axiom	$\langle Adv, 1, 2\rangle$	Adv $\rightarrow$ sometimes	$_1$ sometimes $_2$
3.	Axiom	$\langle V, 2, 3\rangle$	V $\rightarrow$ reads	$_2$ reads $_3$
4.	Axiom	$\langle Det, 3, 4\rangle$	Det $\rightarrow$ the	$_3$ the $_4$
5.	Axiom	$\langle N, 4, 5\rangle$	N $\rightarrow$ paper	$_4$ paper $_5$
6.	Inference	$\frac{\langle Det, 3, 4\rangle\langle N, 4, 5\rangle}{\langle NP, 3, 5\rangle}$	NP $\rightarrow Det\ N$	$_3$ the $_4$ paper $_5$
7.	Inference	$\frac{\langle V, 2, 3\rangle\langle NP, 3, 5\rangle}{\langle VP, 2, 5\rangle}$	VP $\rightarrow V\ NP$	$_2$ reads $_3$ the $_4$ paper $_5$
8.	Inference	$\frac{\langle Adv, 1, 2\rangle\langle VP, 2, 5\rangle}{\langle VP, 1, 5\rangle}$	VP $\rightarrow Adv\ VP$	$_1$ sometimes $_2$ reads $_3$ the $_4$ paper $_5$
9.	Inference	$\frac{\langle NP, 0, 1\rangle\langle VP, 1, 5\rangle}{\langle S, 0, 5\rangle}$	S $\rightarrow NP\ VP$	$_0$ John $_1$ sometimes $_2$ reads $_3$ the $_4$ paper $_5$

Figure 1.6: Example derivation using the CKY CFG parser specified in Figure 1.5 to derive the sentence *John sometimes reads the paper* using the given grammar.

of a node in an elementary tree and the indices that mark the edges of the span dominated by that node. Nodes, notated $\alpha@a \triangleright \ell$, are specified by three pieces of information: α, the identifier of the elementary tree of which the node is a part, a, the address of the node in that tree, and ℓ, the numbered link available at that node or _ if none is available.

We use Gorn addresses to specify the address of the nodes within the trees.[2] Each item has four indices, indicating the left and right edges of the span covered by the node as well as any gap in the node that may be the result of a foot node dominated by the node. Nodes that do not dominate a foot node will have no gap in them. This

[2]The root of a tree has Gorn address ϵ. The j^{th} child of the node with address i has address $i \cdot j$.

is indicated by the use of underscores in place of the indices for the gap. To limit the number of inference rules needed, we define the following function $i \cup j$ for combining indices:[3]

$$i \cup j = \begin{cases} i & j = \_ \\ j & i = \_ \\ i & i = j \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

The side conditions $\mathsf{Init}(\alpha)$ and $\mathsf{Aux}(\alpha)$ hold if α is an initial tree or an auxiliary tree, respectively. $\mathsf{Label}(\alpha@a)$ returns the label of the node in tree α at address a. $\mathsf{Link}(\alpha@a)$ returns the link available at node $\alpha@a$ or null if no link is available. $\mathsf{Foot}(\alpha)$ returns the address of the foot node of an auxiliary tree. $\mathsf{Adj}(\alpha@a \triangleright \ell, \beta)$ determines whether tree β may adjoin into tree α at address a using link ℓ. $\mathsf{Sub}(\alpha@a \triangleright \ell, \beta)$ determines whether tree β may substitute into tree α at address a using link ℓ.

The algorithm traverses the derived tree bottom-up. The Terminal Axiom generates items for the nodes of the elementary trees labeled with the words in the target sentence. The Empty Axiom generates items for frontier nodes of elementary trees that are labeled with the empty string. Because auxiliary trees must be completely parsed in order to satisfy the requirements of the Adjoin inference rule, the Foot Axiom generates items for the feet of auxiliary trees at every possible string position. The Unary and Binary Complete rules move from children to parents within a single elementary tree, respecting the structure of the tree as well as the spans of the items

[3]We also make use of this function for combining other atomic values for which we desire the same behavior.

Item Form:

$$\langle \alpha@a \triangleright \ell, i, j, k, l \rangle$$

Goal:

$$\langle \alpha@\epsilon \triangleright \_, 0, \_, \_, n \rangle \qquad \mathsf{Init}(\alpha)$$

Axioms: $\mathsf{Label}(\alpha@\epsilon) = S$

Terminal Axiom:

$$\langle \alpha@a \triangleright \_, i, \_, \_, i+1 \rangle \qquad \mathsf{Label}(\alpha@a) = w_{i+1}$$

Empty Axiom:

$$\langle \alpha@a \triangleright \_, i, \_, \_, i \rangle \qquad \mathsf{Label}(\alpha@a) = \epsilon$$

Foot Axiom:

$$\langle \alpha@a \triangleright \ell, p, p, q, q \rangle \qquad \mathsf{Aux}(\alpha)$$
$$\mathsf{Foot}(\alpha) = a$$

Inference Rules: $\mathsf{Link}(\alpha@a) = \ell$

Complete Unary:

$$\frac{\langle \alpha@(a \cdot 1) \triangleright \_, i, j, k, l \rangle}{\langle \alpha@a \triangleright \ell, i, j, k, l \rangle} \qquad \alpha@(a \cdot 2) \text{ undefined}$$
$$\mathsf{Link}(\alpha@a) = \ell$$

Complete Binary:

$$\frac{\langle \alpha@(a \cdot 1) \triangleright \_, i, j, k, l \rangle, \langle \alpha@(a \cdot 2) \triangleright \_, l, j', k', m \rangle}{\langle \alpha@a \triangleright \ell, i, j \cup j', k \cup k', m \rangle} \qquad \mathsf{Link}(\alpha@a) = \ell$$

Adjoin:

$$\frac{\langle \beta@\epsilon \triangleright \_, i, p, q, l \rangle, \langle \alpha@a \triangleright \boxed{x}, p, j, k, q \rangle}{\langle \alpha@a \triangleright \_, i, j, k, l \rangle} \qquad \mathsf{Adj}(\alpha@a \triangleright \boxed{x}, \beta)$$

No Adjoin:

$$\frac{\langle \alpha@a \triangleright \boxed{x}, i, j, k, l \rangle}{\langle \alpha@a \triangleright \_, i, j, k, l \rangle}$$

Substitute:

$$\frac{\langle \beta@\epsilon \triangleright \_, i, \_, \_, l \rangle}{\langle \alpha@a \triangleright \_, i, \_, \_, l \rangle} \qquad \mathsf{Link}(\alpha@a) = \boxed{x}$$
$$\mathsf{Sub}(\alpha@a \triangleright \boxed{x}, \beta)$$

Figure 1.7: The CKY algorithm for TAG

they combine. By requiring the antecedent nodes to have null links, these two rules require that any adjunction at the child nodes be complete before the parent node is added to the chart. The Adjoin rule takes an item with an unused link, a span from p to q and a gap from j to k and an item that represents the root of an auxiliary tree and has a gap from p to q and produces an item that is the result of performing the

adjunction operation. The consequent item retains the gap from j to k and the link that was used is no longer available. The No Adjoin rule leaves an item unchanged except for the removal of active link. The Substitute rule takes a completed tree in the antecedent and produces an item corresponding to a substitution node at which that tree can substitute.

This algorithm works in $O(n^6 |G|^2)$ time where n is the length of the string to be parsed and $|G|$ is a measure of the size of the grammar [Vijay-Shanker and Joshi, 1985]. The complexity of the algorithm is easy to compute by looking at the rules of the parser. The n^6 factor comes from the number of independent indices into the string that may appear in any of the inference rules. This corresponds to the number of distinct productions the parser could theoretically need to perform in order to parse a given input string. The measure of grammar size comes from the maximum number of elementary trees that may be represented in the items of any inference rule. The two factors are multiplied together to account for any given elementary trees and any spans of the input string.

A complete listing of the productions of the parser is more overwhelming than instructive, so Figure 1.8 demonstrates a few key productions of the parser when applied to the example sentence *John sometimes reads the paper* using the given grammar. In the chosen examples, the nodes of the elementary trees to which the productions correspond are noted with double pointed arrows. Each consequent serves as proof that it is possible to parse up to the given node in a way that spans the indices indicated in the item and that corresponds to the input sentence. For the Foot Axiom, the foot is inserted at each position at which it might possibly adjoin

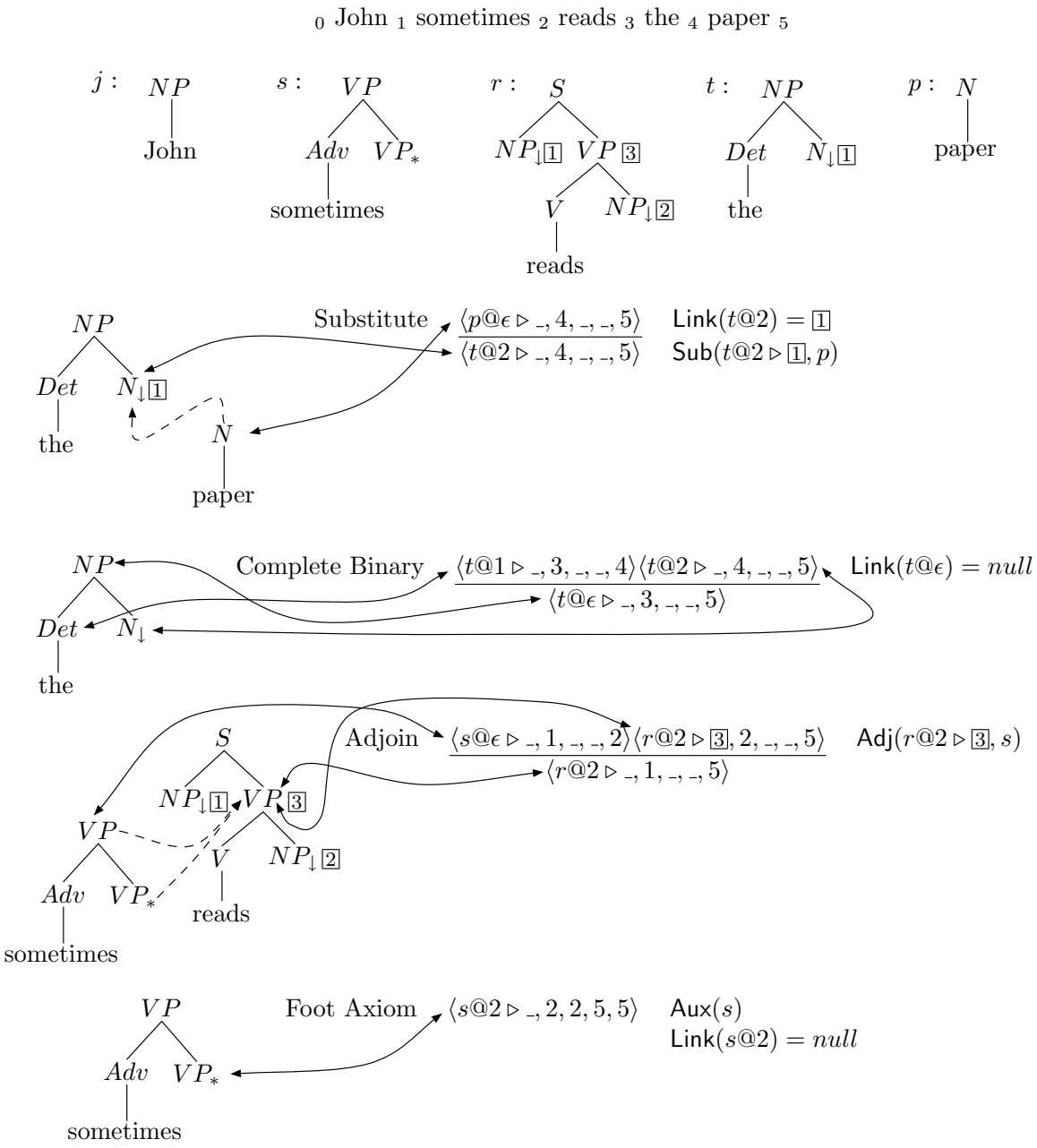


Figure 1.8: A sample TAG for the sentence *John sometimes reads the paper* and a few example productions using the parser given in Figure 1.7.

in the continuation of the parse. The production shown is the one that is actually used in the successful parse of the example sentence. It is possible to constrain the

number of items introduced by this axiom by using the position of any anchor in the auxiliary tree, or by structuring it like the substitution rule so that foot nodes can only be inserted in the chart once there is an adjunction available for the given span.

It is worth noting that although this example does not take advantage of the increase in expressivity provided by the ability of the adjunction operation to "wrap" lexical material around other material, we do make use of this additional expressivity to analyze certain semantic constructions (Chapters 5 and 6). In addition, even without the additional expressivity, the TAG adjunction operation and elementary structures of arbitrary depth allow the CFG-equivalent TAG variant, Tree Insertion Grammar (TIG), to lexicalize CFG grammars without changing the trees produced [Schabes and Waters, 1995]. This allows direct expression of the relationships between lexical items that is not possible using CFG as discussed in Chapter 7.

1.3 Multicomponent TAG

As soon as TAG was defined, its creators offered several multicomponent variants to increase its expressivity [Joshi et al., 1975, Joshi, 1987]. The motivation for these variants derives from the conception of the relationship a TAG is intended to have with the language it characterizes. Each elementary tree is intended to encapsulate a **domain of locality** that encompasses the significant syntactic and semantic relationships into which the lexical item represented by the elementary tree may enter. Although the adjunction operation allows the expansion of the domain of locality of its lexical items beyond what is possible with CFG, certain constructions remain that appear to require a larger domain of locality. For instance, Frank [1992] points out

that it is linguistically sensible for a functional head and a lexical head to be generated in the same derivation step, as in the following example sentences, borrowed from Schuler et al. [2000]:

(3) Does John seem to sleep?

(4) Does John seem likely to sleep?

In these sentences *does* is the functional head of the verb *seem*, making it desirable that the two appear together in a single elementary structure. Because in TAG raising verbs such as *seem* are analyzed as auxiliary trees that adjoin at VP, there is no way to incorporate the functional head into the elementary tree for *seem*.

In order to permit a larger domain of locality, multicomponent TAG allows the elementary structures of the grammar to be sets of trees rather than just single trees. All the trees in a given set must combine with other elementary structures simultaneously, so that they continue to act like a single entity in some respects though they may contribute structure and lexical material to the derived tree at multiple distinct locations. Constraints placed on where an elementary tree set may adjoin create a hierarchy of multicomponent TAG formalisms. In sentence (3), the two trees (one containing the functional head *does* and the other the lexical head *seem*) adjoin to the single elementary tree for *sleep*. However, in sentence (4), *seem* must first adjoin to *likely*. In order for the functional head to have a place to adjoin into *likely*, an additional tree must be added to the elementary set for *likely* and the two trees from the *seem* tree set must adjoin to two different trees in the *likely* tree set. This is shown in Figure 1.9, which is based on a figure from Schuler et al. [2000].

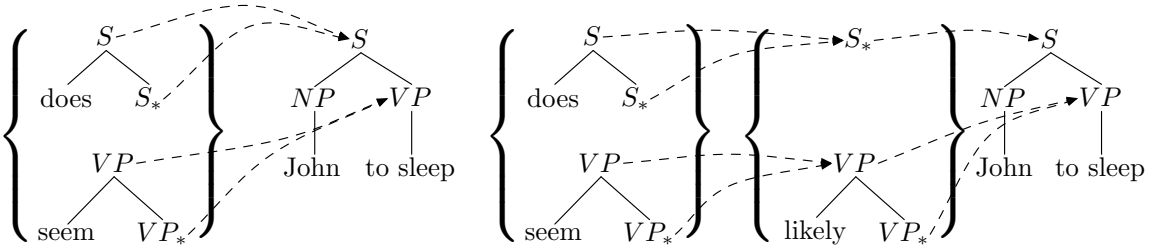


Figure 1.9: Derivations of *Does John seem to sleep?* and *Does John seem likely to sleep*. This example is based on similar figures in Schuler et al. [2000].

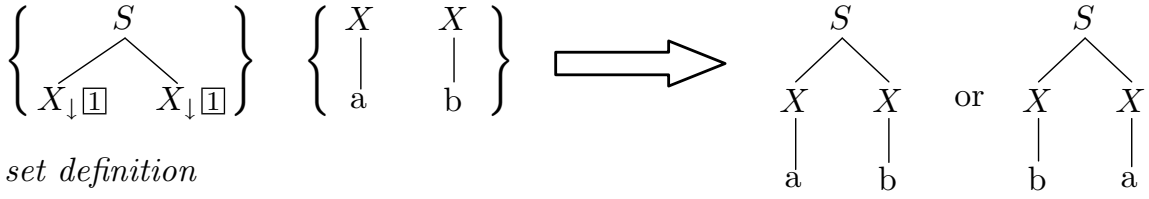


Figure 1.10: An example of the way in which two tree sets may produce several different derived trees when combined under the standard definition of multicomponent TAG.

The first of these adjunctions is said to be **tree-local** because all trees from the *seem* tree set adjoin to a single tree and the second is said to be **set-local** because the trees from the *seem* tree set adjoin to different trees from a given tree set.

Another motivation for the move to multicomponent TAG that is cited in the literature is the encapsulation of coindexed noun phrases within a single elementary trees [Weir, 1988]. This concern is of particular interest for the work on simultaneous syntactic and semantic computation that follows in Chapters 5 and 6 because it combines some arguably syntactic constraints on the distribution of coindexed noun phrases with some clearly semantic information expressed by a coindexation. The problem is particularly clearly presented by wh-questions in which the wh-word itself is introduced in one location in the sentence while the silent noun phrase or trace

with which it is coindexed appears in a structurally distant location.

Unfortunately, the increase in domain of locality and expressivity gained with multicomponent TAG formalisms also increases their complexity. Although multicomponent TAGs have demonstrated benefits in natural language analysis and have been used widely in the literature, various aspects of their computational complexity have not been deeply explored or well-understood. Part I of this thesis attempts to answer some of the unanswered questions about their complexity and then to locate a multicomponent TAG variant that offers both the flexibility and expressivity necessary for linguistic applications and an acceptable level of computational tractability. In the remainder of this section we offer a technical introduction to multicomponent TAG in preparation for the chapters that follow.

Multicomponent TAG (MCTAG) generalizes TAG by allowing the elementary items to be sets of trees rather than single trees [Joshi and Schabes, 1997]. The basic operations are the same but all trees in a set must adjoin (or substitute) into another tree set in a single step in the derivation. To allow for multicomponent adjunction, a numbered link may appear on two or more nodes in a tree, signifying that the adjoining trees must be members of the same tree set. Any tree in a set may adjoin at any link location if it meets other adjunction or substitution conditions such as a matching node label. Thus a single multicomponent link may give rise to many distinct derived trees even when the link is always used by the same multicomponent tree set. An example is given in Figure 1.10. This standard definition of multicomponent adjunction we will call the **set definition**. An alternative and novel definition of multicomponent TAG in which the elementary structures are treated as vectors

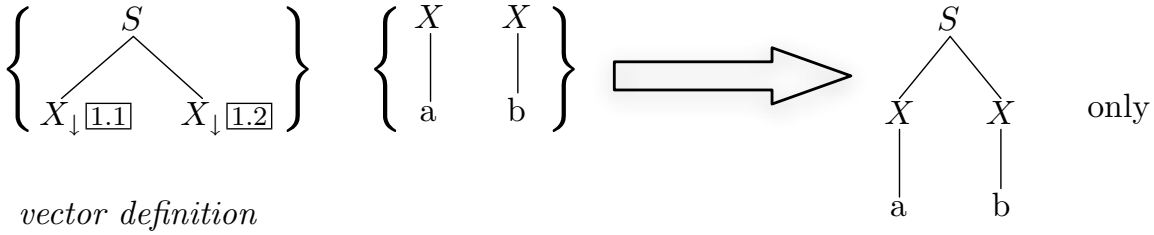


Figure 1.11: An example contrasting the **set definition** of MCTAG (shown in Figure 1.10) with the **vector definition**.

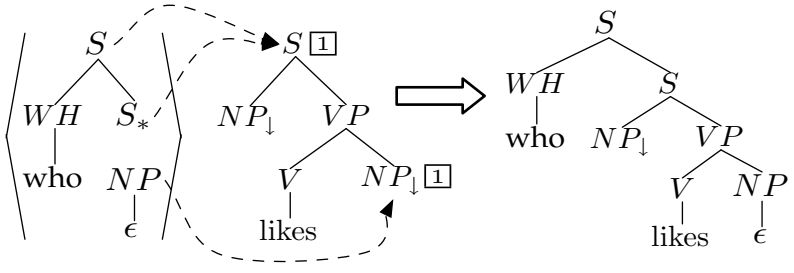


Figure 1.12: An example TL-MCTAG operation demonstrating the use of TL-MCTAG to model wh-question syntax.

is suggested by the explicit use of numbered links at the available adjunction sites. Under this definition, called the **vector definition**, both the locations of a link and the trees in a vector are indexed and the index of a tree and a link location must agree in order for an adjunction to occur. An example contrasting the two definitions is given in Figure 1.11. A derivation tree for a multicomponent TAG is the same as for TAG except that the nodes are labeled with the names of elementary tree sets.

An MCTAG is **tree-local** if all members of a tree set are required to adjoin within a single elementary tree [Weir, 1988]. Using the numbered link notation introduced above for adjunction sites, a tree-local MCTAG (TL-MCTAG) is one in which all locations of a link occur within a single elementary tree. An example TL-MCTAG

operation is given in Figure 1.12. Weir [1988] noted in passing that TL-MCTAG does not increase the generative capacity of TAG; a combination of well-chosen selective and obligatory adjunction constraints and additions of duplicates of trees to the grammar can produce a weakly equivalent TAG. Alternatively, a feature-based TAG where the features enforce the same constraints may be used. Although the generative capacity of the formalism is not increased, conversion from TL-MCTAG to TAG may in some cases require an exponential increase in the size of the grammar as we prove in Chapter 2.

An MCTAG is **set-local** if tree sets are required to adjoin within a single elementary tree set Weir [1988]. Using the numbered link notation, an MCTAG is set-local if all locations of a link occur within a single tree set. Set-local MCTAG (SL-MCTAG) has equivalent expressivity to linear context-free rewriting systems and recognition is provably PSPACE complete, as we will show in Chapter 3.

MCTAG derivation trees are usually written in the same way as TAG derivation trees, except that in the MCTAG case the nodes of the derivation tree represent entire tree sets rather than single trees. This notation is sufficient for TL-MCTAG and SL-MCTAG because of the locality requirements: a tree set can stand in the derivational child relationship to only one other tree set. However, it is worth noting that if the set definition of MCTAG is used, a derivation tree written in this way may not contain sufficient information to disambiguate how a particular tree set makes use of a link. In addition, when an MCTAG is non-local, trees from a single set may adjoin to different tree sets, making this notation insufficient. As an alternative, we introduce the notion of an **elaborated derivation tree** in which each tree from a

tree set appears as a node in a tree. This notion will be helpful in exploring the complexity difference between TL-MCTAG and SL-MCTAG in Chapter 2 and again in defining extensions to TL-MCTAG in Chapter 3.

In multicomponent TAG, multiple adjunction leads to increased expressivity. Because each available adjunction is explicitly notated with a numbered link, our notation implicitly disallows multiple adjunction but permits a third possibility: bounded multiple adjunction. Bounded multiple adjunction permits a specific finite number of adjunctions to occur at a given adjunction site. With bounded multiple adjunction, a formalism may obtain some of the potential linguistic advantages of allowing multiple adjunction while limiting the increase in expressivity and complexity that results from unbounded multiple adjunction [Nesson and Shieber, 2006]. If bounded multiple adjunction is desired, the parsers presented in this thesis may all be modified to allow a list of links at a given node in place of a single link. The spurious ambiguity resulting from the arbitrary choice of which link to use can be avoided by specifying an order over links at a particular location (or links at a particular set of locations) that prevents a link later in the order from being used while a link from earlier in the order remains unused.

1.4 Synchronous TAG

In the linguistic applications addressed in this thesis, pairs of natural language inputs—either the syntax and semantics of a single natural language sentence or the syntactic representations of two sentences of different natural languages that are translations of each other—are related by drawing on the intuition that the structure

of these pairs is shared at some level. That is, although the syntactic structure of two sentences that are translations of each other may be quite divergent, their shared meaning suggests that they may share a derivational structure from which the divergent syntactic structures can be derived.

Focus on TAG as a base formalism for linguistic applications leads to an elegant way to formally characterize this idea. As discussed earlier, in CFG there is no distinction between the derived structures of a grammar and the derivational structures it generates. In TAG, however, the derived structures and the derivation trees need not be isomorphic. This leads to the observation that two derived trees may exhibit quite divergent structure while sharing a single derivation tree. Shieber and Schabes [1990] used this idea as the basis for the definition of synchronous TAG (STAG), a formalism which they among others proposed for use in solving the two problems addressed in Part II of this thesis [Abeillé et al., 1990].

The elementary structures of an STAG are triples of the form $\langle L_i, R_i, \frown_i \rangle$, where L_i is an elementary TAG tree drawn from a TAG grammar G_L, R_i is an elementary TAG tree drawn from a TAG grammar G_R, and $\frown_i$ is a linking relation between nodes of L_i and R_i [Shieber and Schabes, 1990]. The nodes of STAG derivation trees are labeled with the identifier of an elementary structure from the grammar. If γ_1 is the parent of γ_2 in a well-formed STAG derivation tree, the arc between them is labeled with the member of $\frown_1$ that specifies the two nodes from the left and right trees of γ_1 at which the left and right trees of γ_2 adjoin, respectively.

A few examples drawn from early work on STAG serve to demonstrate both the motivation for use of STAG and its operation. Following the proposal of Abeillé and

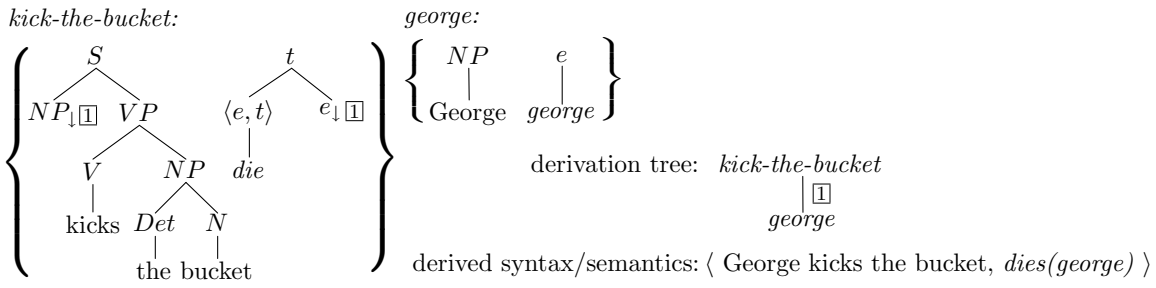


Figure 1.13: Example grammar, derivation tree and derived strings for the sentence *George kicks the bucket* under the idiomatic reading. The nodes of the semantic tree correspond to semantic type. The semantics is read off the tree with the left most terminal node of a branch interpreted as a functor and its siblings to its right interpreted as arguments.

Schabes [1989] that lexicalized TAG is well-suited to modeling idiomatic syntactic constructions, Shieber and Schabes [1990] propose using STAG to extend the modeling to include the semantics of idioms of as well. As exemplified by Figure 1.13, based on an example from Shieber and Schabes [1990], elementary structures for idiomatic readings can easily be incorporated in a lexicon and can also be used to appropriately constrain aspects of the idiomatic construction, such as whether it is frozen or flexible. Extending the formalism to allow synchronization of MCTAG and multiple adjunction, Shieber and Schabes [1990] also propose using it to model quantifiers and quantifier scope ambiguity. We pursue the application of synchronous MCTAG to quantifier scoping in substantial depth in Chapters 5 and 6 and so defer its introduction.

Abeillé et al. [1990] offer several motivating examples for the use of STAG for natural language translation, from which we present two in Figure 1.14. In the first example, the English verb *leave* participates in two distinct elementary tree pairs corresponding to its transitive and intransitive forms. These two forms correspond to

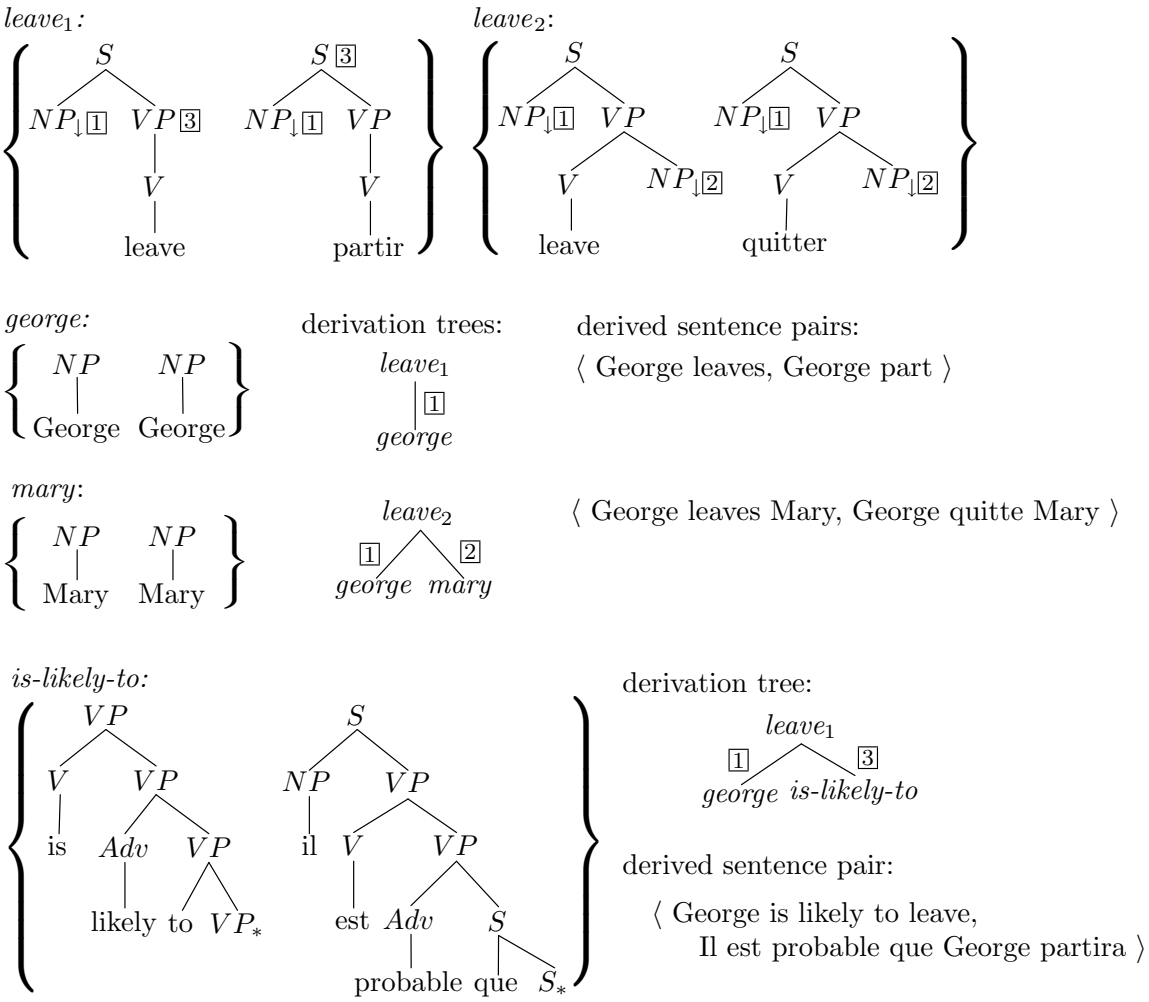


Figure 1.14: Two examples of using STAG to model natural language translation based on examples from Abeillé et al. [1990]. In the first example two different lexical entries for the English verb *leave* in its transitive and intranstive contexts correspond to two different French verbs. In the second example two constructions with different syntactic structures may be paired as translations of each other. Note that in this case we might also choose to further break down these constructions into separate lexical entries for *be/être* and *likely to/probable que*.

distinct verbs, *quitter* and *partir*, respectively, in French. This is modeled easily with STAG. In the second example, the construction *is likely to* in English translates to the French *il est probable que* even though the syntactic structure of the two constructions

differs. An STAG elementary tree pair makes it possible for these divergent structures to be paired and to combine easily with other elementary tree pairs to generate well-formed translations.

Perhaps due to lack of sufficiently deep understanding of the complexity of the underlying formalisms and the complexity introduced by synchronization, research into both of these applications stagnated for some time. In recent years, however, interest in the topic has revived both in the translation and TAG semantics research communities. In machine translation (MT) the desire to incorporate greater syntactic structure into statistical MT (SMT) systems led to the recognition that many current SMT systems can be characterized as synchronous grammars that operate by parsing [Melamed, 2003, 2004]. Because synchronization of even relatively computationally efficient formalisms is NP complete and those underlying formalisms are demonstrably incapable of capturing many correspondences between languages that appear in translation, work in this area has focused on ways to eliminate those aspects of a synchronized grammar that lead to a blow up in parsing complexity [Zhang et al., 2006, Melamed et al., 2004]. Our work in Chapter 4 falls squarely into this line of research, as does the work in the other chapters of Part I when one recognizes the very narrow formal distinction between MCTAG variants and STAG. Chapter 7 describes the design of an SMT system based on a synchronized grammar and reports on the preliminary progress we have made toward its realization.

Although proposed long ago, work in TAG semantics has only been undertaken seriously for the last several years. Due in part to the belief that the TAG syntax derivation trees did not capture the necessary relationships needed for semantic com-

putation, the prevailing line of research in this area does not make use of STAG but rather uses a complex feature-based system in which semantic representations are maintained as features on the derived syntax tree and post-processed into a semantic representation of a sentence once parsing is complete [Kallmeyer and Romero, 2004, Kallmeyer and Scheffler, 2004, Romero et al., 2004, Kallmeyer and Romero, 2007]. The complexity of this method led to our renewed attention to the adequacy of STAG as a formalism for TAG semantics. Chapter 5 presents the results of trying to address the basic aspects of the interface between syntax and semantics that any system must handle. Chapter 6 addresses some of the challenging cases that have been offered in the literature as justifications for why the TAG derivation tree is insufficient as an interface. Although they make no attempt to be comprehensive in their coverage, these chapters are intended to serve as evidence of the potential adequacy and elegance of using STAG to model the interface between natural language syntax and semantics.

1.5 Part 1: Formalisms, Complexity and Algorithms

Part I of this thesis is devoted to the definition and exploration of a variety of TAG variants, including several varieties of MCTAG and STAG. Chapter 2 presents tree-local multicomponent TAG. Prior to the original contributions made by the work in this chapter, the complexity of TL-MCTAG was not well-understood. It was known to have expressivity that was equivalent to TAG but no algorithms for parsing it had been published. In this chapter I present proofs of several original results

demonstrating that TL-MCTAG, even under certain restrictions of the size of the tree sets and the number of links in the tree set, is NP-complete to parse. I then offer the first published algorithm for directly parsing TL-MCTAG and show that it is more efficient than the alternative parsing method derived from an algorithm for parsing multiple context-free grammars. In Chapter 3 I provide original parsing algorithms and, analogously, analyses of complexity, for several restricted variants of TL-MCTAG as well as some new variants defined therein and used in Part II of the thesis. This chapter also discusses SL-MCTAG and explains why the methods applied in the previous chapters are not applicable. Chapter 4 turns to synchronous TAG and provides an original, optimal, and efficient algorithm for transforming an input STAG into an equivalent STAG in which the factor that causes the blow up in parse complexity is minimized. Together, the chapters in Part I of this dissertation give a much fuller and deeper picture of the range of TAG variants, their complexity, and how they may be processed. This serves as the foundation for the choice of formalism to undergird the linguistic applications addressed in Part II.

1.6 Part 2: Linguistic Applications

Part II of this thesis demonstrates the utility of synchronous TAG variants for natural language applications by exploring two particular applications, semantic computation and natural language translation. Chapter 5 defines an STAG-based grammar fragment for English that incorporates both syntax and semantics. It addresses core issues in representing the interface between syntax and semantics, such as constraints on locality and movement and implementation of the binding theory. Chapter 6 tack-

les cases of semantic interpretation that were previously thought to be beyond the limitations of what could be captured using the TAG derivation tree as an interface. The analyses in these chapters are novel extensions of the ideas suggested in early exploration of the use of STAG for semantics and represent a proof-of-concept about the adequacy of STAG for natural language semantics that has led to a renewed interest in its use among researchers in the field [Han, 2006a,b, Frank, 2008]. Chapter 7 gives an overview of research that is still in progress. It presents the design of a statistical machine translation system that, at its core, is based on an STAG in which each side of the grammar is the syntax of a natural language. The novelty of the system is in the idea that a statistically induced substrate that is, at its core, an STAG grammar can then be improved by hybridizing it with elementary tree pairs that have more complex structure and represent linguistically-motivated relationships between the two synchronized languages that cannot be estimated efficiently from a corpus but may be extracted from other sources like bilingual dictionaries or treebanks.

Part I

Formalisms, Complexity and Algorithms

Chapter 2

Tree-Local Multicomponent Tree-Adjoining Grammar

Tree-local multicomponent Tree-Adjoining Grammar (TL-MCTAG) is an appealing formalism for natural language representation because it arguably allows the encapsulation of the appropriate domain of locality within its elementary structures [Kallmeyer and Romero, 2007]. As a result, it has frequently been put to use in a growing body of research into incorporating semantics into the TAG framework [Kallmeyer and Joshi, 2003, Han, 2006b, Nesson and Shieber, 2006, 2007]. Although TL-MCTAG was first introduced by Weir [1988] and shown at that time to be equivalent in expressivity to TAG, the complexity of TL-MCTAG is still not well-understood. Perhaps because of its equivalence to TAG, questions of processing efficiency have not adequately been addressed. This chapter offers a thorough examination of the problem of TL-MCTAG recognition, showing that even highly restricted forms of TL-MCTAG are NP-hard to recognize. However, in spite of the provable difficulty of the recogni-

tion problem, we offer several algorithms that can substantially improve processing efficiency. First, we present a parsing algorithm that improves on the baseline parsing method and runs in polynomial time when both the fan-out—the maximum number of trees in a tree set—and rank—the maximum number of children of a node in a valid derivation tree—of the input grammar are bounded.[1] Second, we offer an optimal, efficient algorithm for factorizing a grammar to produce a strongly-equivalent TL-MCTAG grammar with the rank of the grammar minimized.

2.0.1 Summary of Results

Recent work on the complexity of several TAG variants has demonstrated indirectly that the universal recognition problem for TL-MCTAG is NP-hard [Søgaard et al., 2007]. This result calls into question the practicality of systems that employ TL-MCTAG as the formalism for expressing a natural language grammar. In this chapter we present a more fine-grained analysis of the processing complexity of TL-MCTAG. We demonstrate that even under restricted definitions where either the rank or the fan-out of the grammar is bounded, the universal recognition problem is NP-complete (Section 2.1) .

We also show that the vector definition of TL-MCTAG (a novel variant defined in Chapter 1) is consistent with the linguistic applications of the formalism presented in the literature. Universal recognition of the vector definition of TL-MCTAG is NP-complete when both the rank and fan-out are unbounded. However, when the rank is bounded, the universal recognition problem is polynomial in both the length of the

[1]The terms rank and fan-out are defined in Section 2.1.

input string and the grammar size.

We present a novel parsing algorithm for TL-MCTAG that accommodates both the set and vector definitions of TL-MCTAG (Section 2.3). Although no algorithms for parsing TL-MCTAG have previously been published, the standard method for parsing LCFRS-equivalent formalisms can be applied directly to TL-MCTAG to produce a quite inefficient algorithm in which the polynomial degree of the length of the input string depends on the input grammar. We offer an alternative parser for TL-MCTAG in which the polynomial degree of the length of the input string is constant, though the polynomial degree of the grammar size depends on the input grammar. This alternative parsing algorithm is more appealing than the baseline because it performs universal recognition of TL-MCTAG (vector definition) with constant polynomial degree in both the length of the input string and the grammar size when rank is bounded.

It may not be generally desirable to impose an arbitrary rank bound on TL-MCTAGs to be used for linguistic applications. However, it is possible given a TL-MCTAG to optimally minimize the rank of the grammar. In the penultimate section (Section 2.4) we offer a novel and efficient algorithm for transforming an arbitrary TL-MCTAG into a strongly equivalent TL-MCTAG where the rank is minimized.

In Section 2.5, we conclude with a placement of our results in the context of prior literature.

2.1 Complexity

We present several complexity results for TL-MCTAG. Søgaard et al. [2007] show indirectly that TL-MCTAG membership is NP-complete. For clarity, we present a direct proof here. We then present several novel results demonstrating that the hardness result holds under significant restrictions of the formalism. **Fan-out**, f, measures the number of trees in the largest tree set in the grammar. We show that even when the fan-out is bounded to a maximum of two, the NP-completeness result still holds. The **rank**, r, of a grammar is the largest number of links in any tree in the grammar. We show that when rank is bounded, the NP-completeness result also holds.

A notable aspect of all of the proofs given here is that they do not make use of the additional expressive power provided by the adjunction operation of TAG. Put simply, the trees in the tree sets used in our constructions meet the constraints of Tree Insertion Grammar (TIG), a known context-free–equivalent formalism [Schabes and Waters, 1995]. As a result, we can conclude that the increase in complexity stems from the multicomponent nature of the formalism rather than from the power added by an unconstrained adjunction operation.

2.1.1 Universal Recognition of TL-MCTAG is NP-Complete

In this section we prove that universal recognition of TL-MCTAG is NP-complete when neither the rank nor the fan-out of the grammar is bounded.

Recall the 3SAT decision problem, which is known to be NP-complete. Let $V = \{v_1, \ldots, v_p\}$ be a set of variables and $C = \{c_1, \ldots, c_n\}$ be a set of clauses; each

clause in C is represented by a set of three literals over the alphabet of all literals $L_V = \{v_1, \overline{v}_1, \ldots, v_p, \overline{v}_p\}$. The language 3SAT is defined as the set of all disjunctive formulas over the members of C that are satisfiable.

Theorem 1

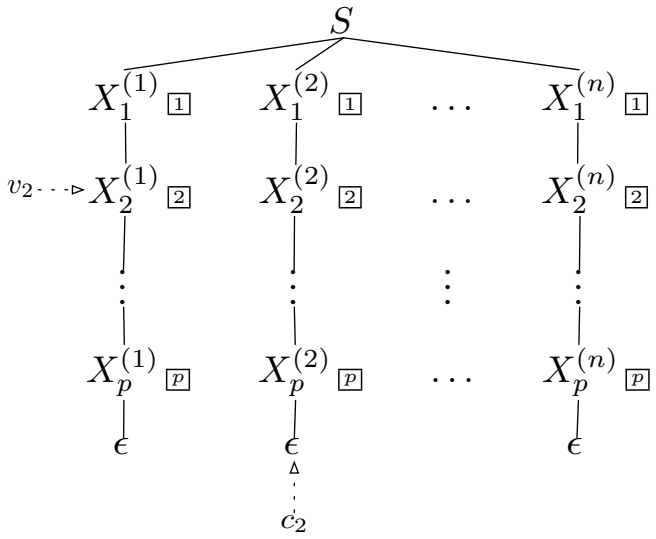
The universal recognition problem for TL-MCTAG with unbounded rank and fan-out is NP-hard.

Proof

Let $\langle V, C \rangle$ be an arbitrary instance of the 3SAT problem.[2] We use the derivations of the grammar to guess the truth assignments for V and use the tree sets to keep track of the dependencies among different clauses in C. Two tree sets are constructed for each variable, one corresponding to an assignment of **true** to the variable and one corresponding to an assignment of **false**. The links in the single initial tree permit only one of these two sets to be used. The tree set for a particular truth assignment for a particular variable v_i makes it possible to introduce, by means of another adjunction, terminal symbols taken from the set $\{1, \ldots, n\}$ that correspond to each clause in C that would be satisfied by the given assignment to v_i. In this way, the string $w = 1 \cdots n$ can be generated if and only if all clauses are satisfied by the truth assignment to some variable they contain.

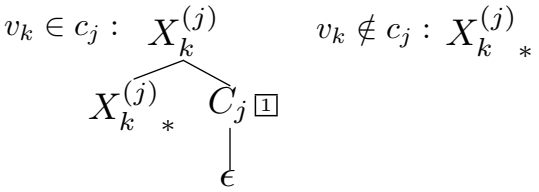
We define a tree-local MCTAG G containing the following tree sets. The initial tree set S contains the single tree:

[2]We follow the proof strategy of Satta and Peserico [2005] in this and the following proof.

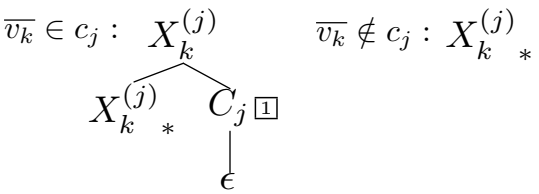


In this tree, the "rows" correspond to the variables and the "columns" to the clauses.

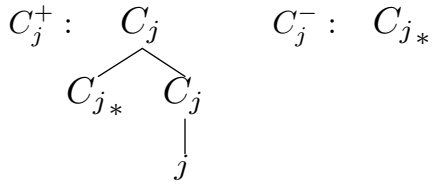
For every variable v_i, $1 \leq i \leq p$, tree set T_i, used when representing an assignment of the value **true** to v_i, contains n trees, one for each clause c_j, $1 \leq j \leq n$, defined as follows:



For every variable v_i, $1 \leq i \leq p$, tree set F_i — used when representing an assignment of the value **false** to v_i — contains n trees, one for each clause c_j, $1 \leq j \leq n$, defined as follows:



For every clause c_j, $1 \leq j \leq n$, tree sets C_j^+ and C_j^- each contain a single tree as shown below. These trees optionally allow the corresponding clause number terminal symbol to be recognized by an appropriate variable instance.



From the definition of G it directly follows that $w \in L(G)$ implies the existence of a truth-assignment that satisfies C. A satisfying truth assignment can be read directly off of any derivation tree for w. If T_i (resp., F_i) is a child of S in the derivation tree, then v_k is **true** (resp., **false**). The converse can be shown by using a satisfying truth assignment for C to construct a derivation for $w \in G$. $\langle G, w \rangle$ can be constructed in deterministic polynomial time because the number of tree sets in the grammar is $2p + 2n + 1$, the total number of trees in the grammar is bounded by $n(2p + 2n + 1)$, and the length of w is n. All trees in the grammar have constant size except for the initial tree, which has size np. ■

Theorem 2

The universal recognition problem for TL-MCTAG with unbounded rank and fan-out is in NP.

Proof

We show that given an arbitrary TL-MCTAG grammar G and any input string w, the determination of $w \in G$ can be performed in non-deterministic polynomial time.

Note that the collection of elementary tree sets of G that can generate the empty

string, $\mathcal{E}$, can be generated in time polynomial in $|G|$ using the standard graph reachability algorithm used for context-free grammars in time polynomial in $|G|$ [Sippu and Soisalon-Soininen, 1988].

We begin by showing that given an arbitrary input string w and derivation tree $\mathcal{D}$ for $w \in G$, there must exist a truncated derivation tree for w that has size no larger than $|G| \cdot |w|$. We define a **truncated derivation tree** as a derivation tree in which the children of elementary tree sets in $\mathcal{E}$ are optionally removed.

Consider $\mathcal{D}$. Each node in $\mathcal{D}$ represents an elementary structure of G: a tuple of one or more TAG trees. We call a node n of $\mathcal{D}$ a **non-splitting node** if a single one of its children in the derivation tree, n_i generates the same lexical material from the input string as n itself.[3] We call it a **splitting node** if more than one of its children generate a non-empty part of the portion of the input string generated by n itself or if n itself contributes lexical material. We proceed from the root of $\mathcal{D}$ examining chains of non-splitting nodes. Assume that the root of $\mathcal{D}$ is a non-splitting node. This means that it has a single child node, n_i that generates the lexical material for the entire input string. Its other children all generate the empty string (and therefore must also be members of $\mathcal{E}$). We truncate the derivation tree at each child of n other than n_i. We now iterate the process on node n_i. If during the examination of a chain of non-splitting nodes we encounter a node identical to one that we have already seen, we remove the entire cycle from the derivation tree because it is not essential to the

[3]The child tree tuple n_i may generate the same lexical material in several distinct pieces which are arranged into the string generated by n when the adjunction occurs. Because the adjunction necessarily connects all of these pieces into a single string in a single predetermined way, it does not matter for our proof that the lexical material derived by the child may be in any order before the adjunctions.

derivation. Because all cycles are removed, the longest possible chain of non-splitting nodes we can find before encountering a splitting node or reaching the bottom of the derivation tree is $|G|$.

If a splitting node is encountered, we truncate all child nodes that generate the empty string and then iterate the process of non-splitting node identification on those children that generate lexical material. In the worst case, the process encounters $w-1$ splitting nodes, each of which may be separated by a chain of non-splitting nodes of maximum length bounded by $|G|$. This process, therefore, produces a truncated derivation tree with size bounded by $|G| \cdot |w|$.

The truncation of the tree at each node that generates the empty string is necessary because the size of the subderivation tree generating the empty string may not be bounded by a polynomial in the size of the grammar. However, the content of the part of the derivation tree used to generate the empty string is not necessary for determining membership of $w \in G$ since we know that each truncated node is a member of $\mathcal{E}$.

To show that TL-MCTAG membership is in NP, we construct a turing machine that will non-deterministically guess a **t**runcated derivation tree of size no larger than $|G| \cdot |w|$. It then checks that the guessed derivation successfully derives w. Because the correctness of the derivation can be checked in linear time, this is sufficient to show that TL-MCTAG membership is in NP. ■

We know from the equivalence of LCFRS and SL-MCTAG (and the rule-to-tree-tuple conversion method used to prove equivalency) [Weir, 1988] and the fact that LCFRS membership is PSPACE-complete that SL-MCTAG membership is also

PSPACE-complete [Kaji et al., 1992, 1994]. Until the above proof it was not known whether TL-MCTAG was in NP. Although the difference in generative capacity between TL-MCTAG and SL-MCTAG is well-known, this proven difference in complexity (assuming NP $\neq$ PSPACE) is novel.

To understand the reason underlying the difference, we note that the bound on the length of non-splitting chains does not hold for set-local MCTAG. In set-local MCTAG a tree tuple may be non-splitting while also performing a permutation of the order of the lexical output generated by its children. Permutation is possible because set-locality allows the tuple of strings generated by a tree tuple to be held separate for an arbitrary number of steps in a derivation. This directly follows the basis of the reasoning of Kaji et al. [1992] in their proof that LCFRS is PSPACE-complete.

2.1.2 Universal Recognition of TL-MCTAG with Bounded Fan-Out is NP-Complete

The grammar constructed in the proof of Theorem 1 has fan-out n, the number of clauses. However, the hardness result proved above holds even if we restrict tree sets to have at most two elements (TL-MCTAG(2)). We use the postfix (2) to indicate the restriction on the fan-out. The result provided here is as tight as possible. If tree sets are restricted to a maximum size of one (TL-MCTAG(1)), the formalism reduces to TAG and the hardness result does not hold.

Theorem 3

The universal recognition problem for TL-MCTAG(2) with fan-out limited to two and unbounded rank is NP-complete.

Proof

Let $\langle V, C\rangle$ be an arbitrary instance of the 3SAT problem. We define a more complex string $w = w^{(1)}w^{(2)}\cdots w^{(p)}w_c$ where w_c is a representation of C and $w^{(i)}$ controls the truth assignment for the variable v_i, $1 \leq i \leq p$. Then we construct a TL-MCTAG(2) grammar G such that each $w^{(i)}$ can be derived from G in exactly two ways using the left members of tree sets of size 2 that correspond to the variables (and a single initial tree set of size 1). The form of the prefix string enforces the constraint of permitting only two derivations by requiring a strictly alternating string of terminal symbols that can only be generated by the grammar when the truth assignment is stable for a particular variable. The derivation of the prefix string $w^{(1)}w^{(2)}\cdots w^{(p)}$ therefore corresponds to a guess of a truth assignment for V. The right trees from the tree sets derive the components of w_c that are compatible with the guessed truth assignments for $v_1, \ldots, v_p$. Below we explain how $\langle G, w\rangle$ is constructed given an instance of 3SAT $\langle V, C\rangle$.

For every variable v_i, $1 \leq i \leq p$, let $\mathcal{A}_i = \{c_j \mid v_i \in c_j\}$ and $\overline{\mathcal{A}}_i = \{c_j \mid \overline{v}_i \in c_j\}$ be the sets of clauses in which v_i occurs positively and negatively, respectively; let also $m_i = |\mathcal{A}_i| + |\overline{\mathcal{A}}_i|$ be the number of occurrences of the variable v_i. Let $\Sigma' = \{a_i, b_i \mid 1 \leq i \leq p\}$ be an alphabet of not already used symbols; let $w^{(i)}$ (again for $1 \leq i \leq p$) denote a sequence of $m_i + 1$ alternating symbols a_i and b_i such that if m_i is even $w^{(i)} = (a_ib_i)^{m_i/2}a_i$ and if m_i is odd $w^{(i)} = (a_ib_i)^{(m_i+1)/2}$. We define three functions,

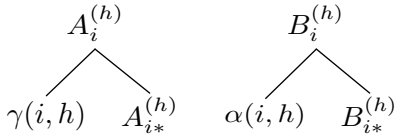
α, γ, and $\overline{\gamma}$ to aid in the construction. The functions γ and $\overline{\gamma}$ are used to produce pieces of the prefix string and will only produce the correct prefix string for a variable if the truth assignment is consistent within the derivation. The function α is used to produce strings representing the clauses satisfied by a particular truth assignment to a variable. For every variable v_i, $1 \leq i \leq p$, the clauses $\alpha(i,1), \alpha(i,2), \ldots, \alpha(i,|\mathcal{A}_i|)$ are all the clauses in $\mathcal{A}_i$ and the clauses $\alpha(i,|\mathcal{A}_i|+1), \ldots, \alpha(i,m_i)$ are all the clauses in $\overline{\mathcal{A}}_i$. Further, for every $1 \leq i \leq p$, let $\gamma(i,1) = a_i b_i$ and let $\gamma(i,h) = a_i$ if h is even and $\gamma(i,h) = b_i$ if h is odd, for $2 \leq h \leq m_i$. For every $1 \leq i \leq p$, let $\overline{\gamma}(i,h) = a_i$ if h is odd, and $\overline{\gamma}(i,h) = b_i$ if h is even for $1 \leq h \leq m_i - 1$ and let $\overline{\gamma}(i,m_i) = a_i b_i$ if m_i is odd and $b_i a_i$ if m_i is even. The crucial property of γ and $\overline{\gamma}$ is that a string $w^{(i)}$ can be parsed either as a sequence of $\gamma(i,\cdot)$ or $\overline{\gamma}(i,\cdot)$ strings, not intermixed elements. The grammar must "commit" to parsing the string one way or the other, corresponding to committing to a value for the variable v_i.

We define a TL-MCTAG(2) G to consist of the tree sets described below. We construct a tree set of length two for each combination of a variable and clause that the variable can satisfy under some truth assignment, filler tree sets of length two that only contribute the string indicating the truth assignment of the variable but no satisfied clause, and a singleton tree set containing only an initial tree rooted in S. The initial tree has $n+1$ branches with the first branch intended to yield the prefix string $w^{(1)} \cdots w^{(p)}$ and the $(k+1)$-st branch intended to yield c_k where $1 \leq k \leq n$. Although it is possible to generate strings not of the form of w using this construction, given a pair $\langle G, w \rangle$ where w respects the definition above, we show that $w \in L(G)$ if and only if C is satisfiable.

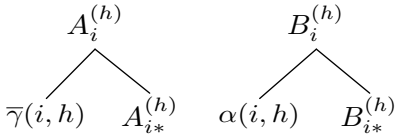
The initial tree set S contains the single tree pictured in Figure 2.1.[4] The name of each link in the initial tree set is composed of three indices that indicate the role of the link. The first index, i, corresponds to variable v_i. The second is an index into the series $1 \cdots m_i$ where m_i is defined from v_i as described above. The third index, j, corresponds to a clause c_j. The use of multiple indices to name the links is for clarity only. They may be renamed freely.

For every variable v_i, $1 \leq i \leq p$, and index h, $1 \leq h \leq m_i$:

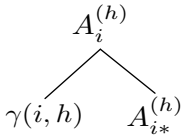
- if $h \leq |\mathcal{A}_i|$, tree set $T_i^{(h)+}$ contains the following two trees:



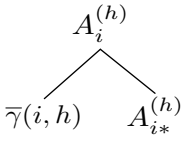
- if $h > |\mathcal{A}_i|$, tree set $F_i^{(h)+}$ contains the the following two trees:



- for all h, tree set $T_i^{(h)-}$ contains the following two trees:



- for all h, tree set $F_i^{(h)-}$ contains the following two trees:



[4] Although we permit the presence of multiple links at a single node in the S tree, as noted earlier we follow the usual TAG convention of disallowing multiple adjunction. If one of the links at a node is used, the other links at that node are assumed to be unavailable in the derivation.

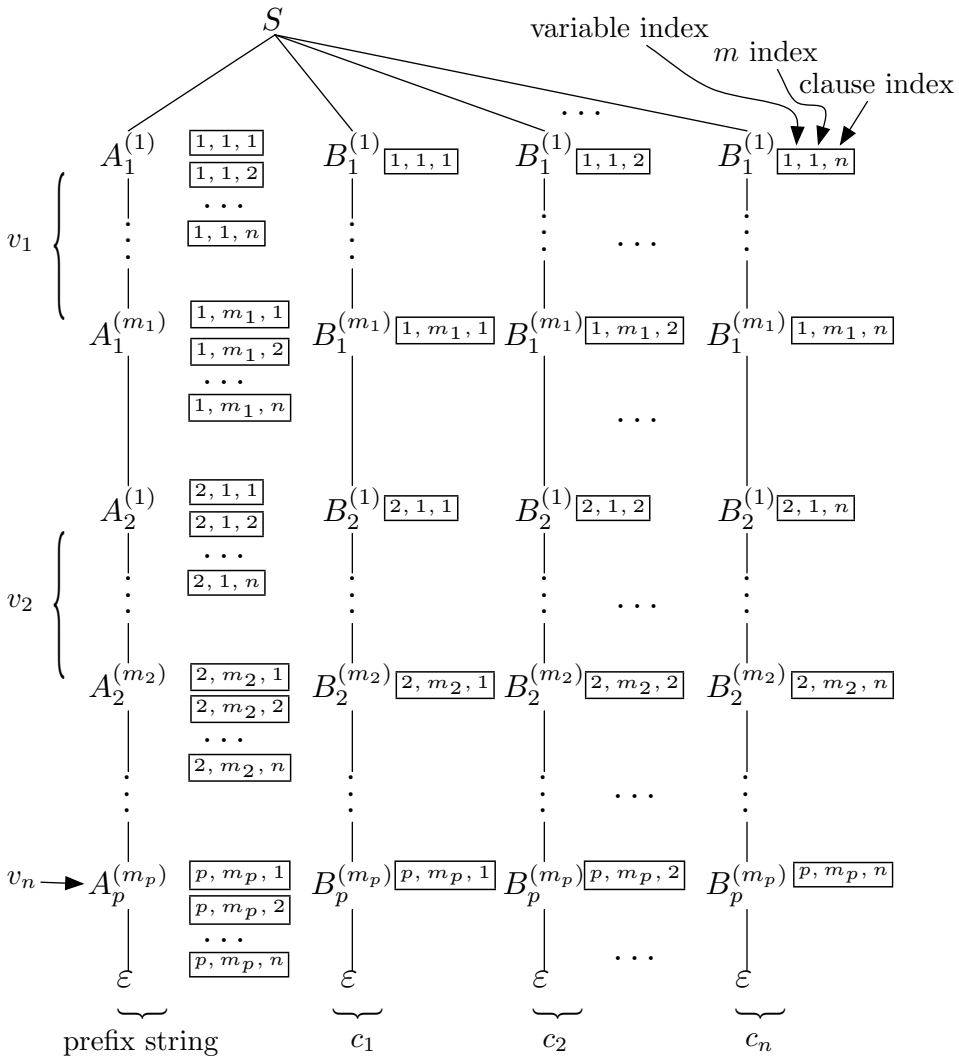


Figure 2.1: The start tree for TL-MCTAG(2) grammar G. The multiply-indexed link numbers are for clarity only and are treated as simple link names.

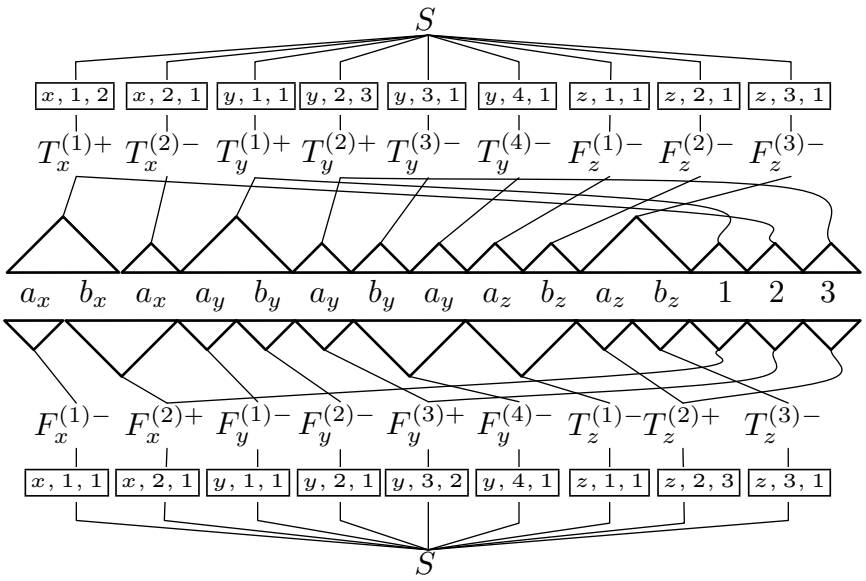


Figure 2.2: Example derivations of two satisfying assignments for the boolean formula $(\overline{x} \vee y \vee z) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (y \vee \overline{y} \vee z)$.

An illustrative example is provided in Figure 2.2. In this example we demonstrate derivations of two possible satisfying truth assignments for boolean formula $(\overline{x} \vee y \vee z) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (y \vee \overline{y} \vee z)$. The truth assignments correspond to whether the T or F tree sets are used in the derivation of the prefix string for a particular variable. As can be seen from the example, the structure of the prefix string enforces that either all T tree sets or all F tree sets are chosen for a particular variable. Each tree set marked with a + is used to satisfy a single clause. Which clause a tree set satisfies can be read off the link number at which it adjoins.

$|G|$ and $|w|$ are polynomially related to p and n. First note that the sum of all the m_i is maximally $3n$ because only $3n$ variables may contribute to the satisfaction of C. There are three tree sets corresponding to each variable and clause that it participates in, for a total of no more than $9pn$ tree sets. With the single initial

tree, the total number of tree sets is bounded by $9pn + 1$. Since every tree set other than the initial tree has two trees in it, the total number of trees in the grammar is bounded by $18pn + 1$. The size of all trees other than the initial tree is bounded by a constant. The size of the initial tree is bounded by $3pn$. The length of w is bounded by $4n + p$.

From a derivation of $w \in L(G)$ we can find a truth assignment satisfying C by examining the derivation. If the tree sets $T_i^{(h)+}$ or $T_i^{(h)-}$ are children of S for some i and all h where $1 \leq i \leq p$ and $1 \leq h \leq m_i$, then v_i is true. If the tree sets $F_i^{(h)+}$ or $F_i^{(h)-}$ are children of S for some i and all h where $1 \leq i \leq p$ and $1 \leq h \leq m_i$ then v_i is false. By the construction, if w is of the form described above, for a given variable v_i only two derivations of $w^{(i)}$ will be possible, one in which all tree sets corresponding to that variable are T tree sets and one in which all are F tree sets. Starting from a truth assignment that satisfies C, we can prove $w \in L(G)$ by induction on $|V|$.

That this problem is in NP can be seen from the same reasoning as in the proof of Theorem 2. ■

2.1.3 Universal Recognition of TL-MCTAG with bounded rank is NP-Complete

We now show that universal recognition of TL-MCTAG is NP-complete even when the rank is bounded.

We briefly recall here the definition of a decision problem called **3-partition**. Let t and $s_i \leq t$ be positive integers, $1 \leq i \leq 3m$, $m \geq 1$. The language 3PAR is defined as the set of all tuples $\langle s_1, \ldots, s_{3m}, t \rangle$, satisfying the following condition: the multiset

$Q = \{s_1, \ldots, s_{3m}\}$ can be partitioned into multisets Q_i, $1 \leq i \leq m$, such that $|Q_i| = 3$ and $\sum_{s \in Q_i} s = t$.

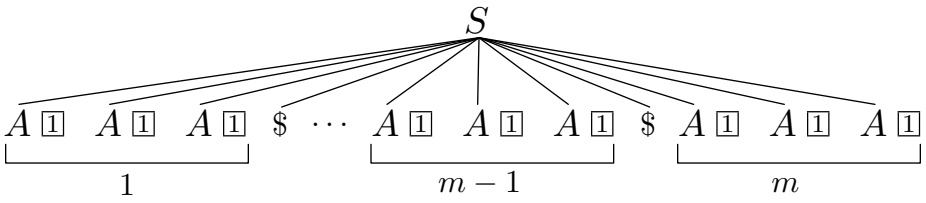
Language 3PAR is strongly NP-complete [Garey and Johnson, 1979]. This means that 3PAR is NP-complete even in case the integers s_i are all represented in unary notation.

Theorem 4

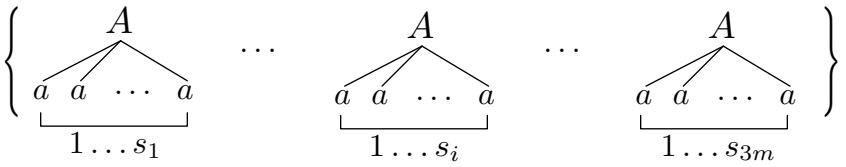
The universal recognition problem for TL-MCTAG with rank 1 and unbounded fan-out is NP-complete.

Proof

We provide a reduction from 3PAR. Let $\langle s_1, \ldots, s_{3m}, t \rangle$ be an input instance of the 3-partition problem, with all of the integers s_i represented in unary notation. Our target grammar G is defined as follows. We use a set of nonterminal symbols $\{S, A\}$, with S the start symbol. We also use a set of terminal symbols $\{a, \$\}$. G contains two elementary tree sets. The first set has a single elementary tree γ, corresponding to a context-free production of the form $S \rightarrow (AAA\$)^{m-1}AAA$:



Tree γ has a unique link impinging on all of the $3m$ occurrences of nonterminal A. The second (multi)set of G contains elementary trees γ_i , $1 \leq i \leq 3m$. Each γ_i corresponds to a context-free production of the form $A \rightarrow a^{s_i}$:



We also construct a string $w = (a^t\$)^{m-1}a^t$.

If there exists a partition for multiset $Q = \{s_1, \ldots, s_{3m}\}$ satisfying the 3PAR requirement, we can directly construct a derivation for w in G, by sorting the elementary trees in the second set accordingly, and by inserting these trees into the link of the elementary tree γ. Conversely, from any derivation of w in G, we can read off a partition for Q satisfying the requirement for membership in 3PAR for the input instance of the 3-partition problem.

Finally, it is easy to see that G and w can be constructed in linear deterministic time with respect to the size of the input instance of the 3-partition problem.

That this problem is in NP can be seen from the same reasoning as in the proof of Theorem 2. ■

2.1.4 Universal Recognition of TL-MCTAG with Fixed Input String is NP-Complete

We now show the unusual complexity result that universal recognition of TL-MCTAG is NP-complete even when the input string is fixed. Although it is uncommon to require this result, we rely on it in Section 2.3 to demonstrate that our parser has better time complexity than the baseline parsing method for TL-MCTAG that we generalize from the standard parsing method for LCFRS-equivalent formalisms.

We reduce from a variant of the 3-SAT problem introduced above in which each variable occurs in at most four clauses with no repeats in a clause. This problem was shown to be NP-complete by Tovey [1984].

Theorem 5

Universal recognition of TL-MCTAG is NP-complete when the input string is fixed.

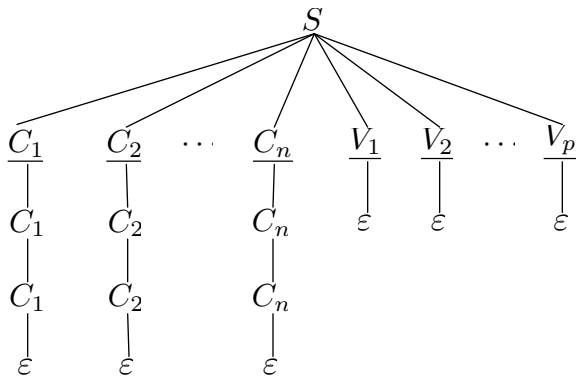
Proof

Let $\langle V, C \rangle$ be an arbitrary instance of the 3SAT problem where each variable occurs in no more than four clauses and does not repeat within a single clause. As in the proof of Theorem 1, we use the derivations of the grammar to guess the truth assignments for V and use the tree sets to keep track of the dependencies among different clauses in C. Two tree sets are constructed for each variable, one corresponding to a true assignment and one corresponding to a false assignment. The prohibition on multiple adjunction ensures that only one of these two tree sets can be used for each variable. The tree set for a particular truth assignment for a particular variable v_i makes it possible to satisfy obligatory adjunction constraints for nonterminal symbols that represent each of the clauses v_i satisfies in the 3-SAT formula.[5] Additional adjunction sites for each clause provide overflow space in the event that more than one variable satisfies a particular clause. We fix the input string w to be the empty string. None of the trees of the grammar contain any terminal symbols. However, a successful parse

[5]Obligatory adjunction constraints are standard in the definition of TAG and MCTAG [Weir, 1988]. However, obligatory adjunction may be avoided in this proof by creating a larger grammar in which a separate tree set is created for each combination of clauses that may be satisfied by a given variable. Because each variable may appear in no more than four clauses, this increases the number of tree sets in the grammar by 2^4. We leave the details of this alternative proof strategy to the reader.

of the empty string can only be achieved if all of the obligatory adjunction constraints are satisfied and this occurs if and only if all clauses of the formula are satisfied by the truth assignment to some variable.

We define a tree-local MCTAG G containing the following tree sets. We notate obligatory adjunction constraints by underlining the nodes at which they apply. The initial tree set S contains the single tree:



For every variable v_i, $1 \leq i \leq p$, tree set T_i (resp. F_i), used when representing an assignment of the value **true** (resp. **false**) to v_i, contains at most five trees, one for the variable itself and one for each clause c_j, $1 \leq j \leq n$, such that when v_i is true (resp. false) c_j is satisfied. More formally, tree set T_i contains trees V_{i*} and C_{j*} if and only if $v_i \in c_j$, for $1 \leq j \leq n$. Tree set F_i contains trees V_{i*} and C_{j*} if and only if $\overline{v_i} \in c_j$, for $1 \leq j \leq n$.

Note that the diagram of the initial tree does not show the explicitly notated link locations that we use throughout the chapter. We omit the link locations to avoid cluttering the diagram. However, because each variable occurs at most four times in the formula, the total number of links is bounded by pn^{12}.

From the definition of G it directly follows that $\varepsilon \in L(G)$ implies the existence of a

truth-assignment that satisfies C. A satisfying truth assignment can be read directly off of any derivation tree for w. If T_i (resp., F_i) is a child of S in the derivation tree, then v_k is **true** (resp., **false**). The converse can be shown by using a satisfying truth assignment for C to construct a derivation for $w \in G$.

$\langle G, w \rangle$ can be constructed in deterministic polynomial time because the number of tree sets in the grammar is $2p + 1$, the total number of trees in the grammar is bounded by $n(2p + 1)$, and the length of w is 0. All trees in the grammar have constant size except for the initial tree, which has size $3n + p$.

That this problem is in NP can be seen from the same reasoning as in the proof of Theorem 2.[6] ■

2.2 TL-MCTAG with Tree Vectors

The proof of NP-hardness of TL-MCTAG in the bounded rank case given above (Theorem 4) depends crucially on the treatment of the elementary structures of the TL-MCTAG as unordered sets. In order to produce the satisfying partitions for the 3-partition problem, any tree from the second tree set must be able to adjoin at any location of link 1 in the first tree set. This is in accordance with the usual definition of MCTAG.

The dependence of our bounded-rank proof on the set definition of TL-MCTAG does not in itself show that vector definition TL-MCTAG is polynomial in the bounded

[6]The use of obligatory adjunction constraints complicates this reasoning because it is not possible to truncate a branch of the derivation tree that contains an obligatory adjunction. However, if we omit the obligatory adjunction constraints and instead use an expanded (but still polynomial size) grammar, as described in footnote 5, the proof that recognition is NP goes through without modification.

rank case. We show this constructively in Section 2.3 by presenting a parser for vector definition TL-MCTAG for which the polynomial degree of both the length of the input string and the grammar size is constant when the rank of the input grammar is bounded.

The difference in complexity between the set and vector definitions of TL-MCTAG makes use of the vector definition an appealing possibility for research using TL-MCTAG for natural language applications. Although all uses of TL-MCTAG in the computational linguistics literature assume the set definition of TL-MCTAG, the linguistic analyses therein do not require the additional flexibility provided by the set definition. In every case of which we are aware, the link locations used by the multicomponent tree sets are completely determined by the constraints imposed by the node labels at the adjunction and substitution sites. As a result, these grammars may be converted to the vector definition without any change in generative capacity or any increase in grammar size but with crucial gains in processing efficiency.

2.3 Parsing

Although no algorithms for parsing TL-MCTAG have previously been published, the standard method for parsing LCFRS-equivalent formalisms can be applied directly to TL-MCTAG to produce an algorithm for which the polynomial degree of both the length of the input string and the grammar size depends on the input grammar. We offer a novel parser for TL-MCTAG for which the polynomial degree of the length of the input string is constant. For the set definition of TL-MCTAG the polynomial degree of the grammar size depends on both the rank and fan out of the input

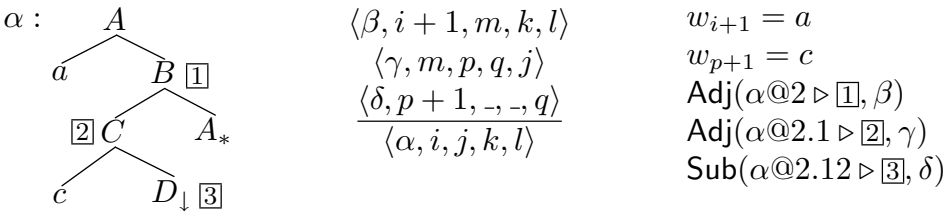


Figure 2.3: The deductive rule generated for tree α using the naive TAG parsing method.

grammar. For the vector definition of TL-MCTAG the polynomial degree of the grammar size depends on the rank of the input grammar but contains no index of the fan out.

We begin with some additional comments on TAG parsing before introducing our novel parsing algorithm.

2.3.1 TAG Parsing

Following the method of Seki et al. [1991], a naive parser for TAG may be constructed by generating a single inference rule for each tree in the grammar. For a tree containing r links, the rule will have r antecedents with each antecedent item representing a tree that can adjoin at one of the links. Each adjoining tree will cover a span of the input string that can be represented by four indices, indicating the left and right edges of the span and of the subspan that will ultimately be dominated by its foot node. Because the location of the links within the consequent tree is known, the indices in the antecedent items are not entirely independent. An example is given in Figure 2.3. Observation shows that there will be a worst case of $2(r + 1)$ independent indices in a given rule. Since each adjoining tree is independent, there may be

$r + 1$ different trees represented in a single rule. This results in a time complexity of $O(n^{2(r+1)} |G|^{r+1})$ where n is the length of the input string, $|G|$ is a representation of the grammar size, and r is the rank of the input grammar.

Following Graham et al. [1980] in their optimization of the Earley parser [Earley, 1970], the identifiers of specific trees need not be represented in the items of the parser. Rather the tree identifiers may be replaced by the labels of the root nodes of those trees, effectively bundling items of trees that share a root node label and cover the same span. This modification to the algorithm reduces the time complexity of the parser to $O(n^{2(r+1)} |G|)$. We refer to this method of reducing complexity by removing unnecessary information about specific elementary structures from the items of the parser as the GHR optimization. When applied it reduces the time complexity in the grammar size but does not alter the basic form of the time complexity expression. It remains a single term consisting of the product of a polynomial in the input string length and a polynomial in the grammar size. We will return to this observation when examining the complexity of TL-MCTAG parsing.

Shieber et al. [1995] and Vijay-Shanker [1987] apply the Cocke-Kasami-Younger (CKY) algorithm first introduced for use with context-free grammars in Chomsky normal form [Kasami, 1965, Younger, 1967] to the TAG parsing problem to generate parsers with a time complexity of $O(n^6 |G|^2)$. The speed up in the parser comes from traversing elementary trees bottom up, handling only one link at a time. As a result, no inference rule needs to maintain information about more than one link at a time. If the GHR optimization is applied, the time complexity is reduced to $O(n^6 |G|)$. The inference rules for this parsing algorithm are provided in Chapter 1.

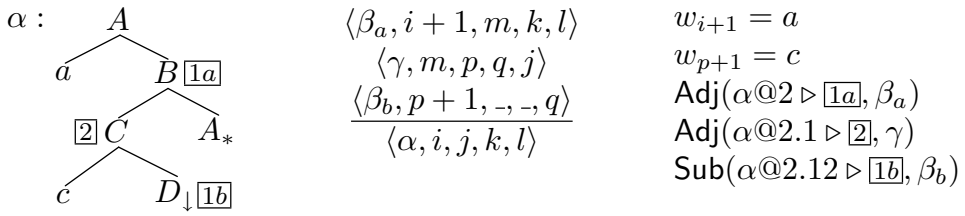


Figure 2.4: The deductive rule generated for tree α_i using the naive TL-MCTAG parsing method.

2.3.2 CKY-Style Tree-Local MCTAG Parsing

As shown in Figure 2.4, the naive algorithm for parsing TAG may also be applied to TL-MCTAG. The only difference is that each link may have multiple locations within a given tree. Let r and f represent the rank and fan-out of the input grammar, respectively. The time complexity of the naive parser will therefore be $O(n^{2(rf+1)}\,|G|^{r+1})$. However, the GHR optimization cannot straightforwardly be applied because maintenance of tree locality requires items to carry information about the identities of the specific trees involved rather than just the labels of the root nodes. Theorem 5 addresses the case in which the input string length is 0. Therefore, in this case, any factor in the complexity including the input string length cannot contribute to the overall time complexity. By showing that the problem is NP-complete when the input string length is 0, Theorem 5 demonstrates that there must be some exponential factor or term in the time complexity expression other than the input string length factor. Due to the earlier observation that the GHR optimization does not change the form of the time complexity expression, Theorem 5 therefore shows that the GHR optimization cannot reduce the exponent of the grammar size term to a constant unless $P = NP$. This leaves open the possibility of the existence of an algorithm that is

polynomial in the grammar size but has an additional exponential term in the time complexity expression. However, such an algorithm, if it exists, cannot be generated by application of the GHR optimization to the baseline parser.

Item Form:

$$\langle \alpha_x @ a \triangleright \ell, i, j, k, l, \Lambda \rangle$$

Goal Item:

$$\langle \alpha_1 @ \varepsilon \triangleright \_, 0, \_, \_, n, \emptyset \rangle \qquad \begin{array}{l} \mathsf{Init}(\alpha_1) \\ \mathsf{Label}(\alpha_1 @ \varepsilon) = S \\ |\alpha| = 1 \end{array}$$

Axioms:

Terminal Axiom

$$\langle \alpha_x @ a \triangleright \_, i, \_, \_, i+1, \emptyset \rangle \qquad \mathsf{Label}(\alpha_x @ a) = w_{i+1}$$

Empty Axiom

$$\langle \alpha_x @ a \triangleright \_, i, \_, \_, i, \emptyset \rangle \qquad \mathsf{Label}(\alpha_x @ a) = \varepsilon$$

Foot Axiom

$$\langle \alpha_x @ \mathsf{Ft}(\alpha_x) \triangleright \ell, p, p, q, q, \emptyset \rangle \qquad \begin{array}{l} \mathsf{Aux}(\alpha_x) \\ \mathsf{Link}(\alpha_x @ \mathsf{Ft}(\alpha_x)) = \ell \end{array}$$

Figure 2.5: Modified item form, goal, and axioms for the CKY algorithm for tree-local MCTAG. Inference rules of the algorithm are given in Figure 2.6.

We can improve on the naive algorithm by generalizing the CKY TAG parsing algorithm presented above to the TL-MCTAG case. The direct specification of a CKY-style tree-local MCTAG parser is given in Figures 2.5 and 2.6. For a tree set or vector α from G, we notate the trees in the set or vector using indices that are indicated as subscripts on the tree set identifier. A tree set or vector α from G with length two will therefore contain trees α_1 and α_2. Under the set definition these indices serve only as a way of differentiating the members of the tree set. Under the vector definition, the index must match the index of the link location where the tree

Inference Rules:

Unary Complete

$$\frac{\langle \alpha_x@(a \cdot 1) \triangleright \_, i, j, k, l, \Lambda \rangle}{\langle \alpha_x@a \triangleright \ell, i, j, k, l, \Lambda \rangle}$$

$\alpha_x@(a \cdot 2)$ undefined
$\mathsf{Link}(\alpha_x@a) = \ell$

Binary Complete

$$\frac{\langle \alpha_x@(a \cdot 1) \triangleright \_, i, j, k, l, \Lambda_1 \rangle \langle \alpha_x@(a \cdot 2) \triangleright \_, l, j', k', m, \Lambda_2 \rangle}{\langle \alpha_x@a \triangleright \ell, i, j \cup j', k \cup k', m, \Lambda \rangle}$$

$\mathsf{Link}(\alpha_x@a) = \ell$
$\mathsf{Valid}(\Lambda_1 \cup \Lambda_2)$
$\mathsf{Filter}(\Lambda_1 \cup \Lambda_2, \alpha_x@a \triangleright \ell) = \Lambda$

Adjoin (**set definition**):

$$\frac{\langle \beta_y@\varepsilon \triangleright \_, i, p, q, l, \emptyset \rangle \langle \alpha_x@a \triangleright \boxed{\sigma}_z, p, j, k, q, \Lambda_1 \rangle}{\langle \alpha_x@a \triangleright \_, i, j, k, l, \Lambda \rangle}$$

$\mathsf{Adj}(\alpha_x@a \triangleright \boxed{\sigma}_z, \beta_y)$
$\mathsf{Valid}(\Lambda_1 \cup \{\boxed{\sigma}_z \mapsto \beta_y\})$
$\mathsf{Filter}(\Lambda_1 \cup \{\boxed{\sigma}_z \mapsto \beta_y\}, \alpha_x@a \triangleright \_) = \Lambda$

Adjoin (**vector definition**):

$$\frac{\langle \beta_y@\varepsilon \triangleright \_, i, p, q, l, \emptyset \rangle \langle \alpha_x@a \triangleright \boxed{\sigma}_y, p, j, k, q, \Lambda_1 \rangle}{\langle \alpha_x@a \triangleright \_, i, j, k, l, \Lambda \rangle}$$

$\mathsf{Adj}(\alpha_x@a \triangleright \boxed{\sigma}_y, \beta_y)$
$\mathsf{Valid}(\Lambda_1 \cup \{\boxed{\sigma} \mapsto \beta\})$
$\mathsf{Filter}(\Lambda_1 \cup \{\boxed{\sigma} \mapsto \beta\}, \alpha_x@a \triangleright \_) = \Lambda$

Substitute (**set definition**):

$$\frac{\langle \beta_y@\varepsilon \triangleright \_, i, \_, \_, l, \emptyset \rangle}{\langle \alpha_x@a \triangleright \_, i, \_, \_, l, \Lambda \rangle}$$

$\mathsf{Link}(\alpha_x@a) = \boxed{\sigma}_z$
$\mathsf{Subst}(\alpha_x@a \triangleright \boxed{\sigma}_z, \beta_y)$
$\mathsf{Filter}(\{\boxed{\sigma}_z \mapsto \beta_y\}, \alpha_x@a \triangleright \_) = \Lambda$

Substitute (**vector definition**):

$$\frac{\langle \beta_y@\varepsilon \triangleright \_, i, \_, \_, l, \emptyset \rangle}{\langle \alpha_x@a \triangleright \_, i, \_, \_, l, \Lambda \rangle}$$

$\mathsf{Link}(\alpha_x@a) = \boxed{\sigma}_y$
$\mathsf{Subst}(\alpha_x@a \triangleright \boxed{\sigma}_y, \beta_y)$
$\mathsf{Filter}(\{\boxed{\sigma} \mapsto \beta\}, \alpha_x@a \triangleright \_) = \Lambda$

No Adjoin (**set definition**):

$$\frac{\langle \alpha_x@a \triangleright \boxed{\sigma}_y, i, j, k, l, \Lambda_1 \rangle}{\langle \alpha_x@a \triangleright \_, i, j, k, l, \Lambda \rangle}$$

$\mathsf{Valid}(\Lambda_1 \cup \{\boxed{\sigma}_y \mapsto \mathtt{na}_y\})$
$\mathsf{Filter}(\Lambda_1 \cup \{\boxed{\sigma}_y \mapsto \mathtt{na}_y\}, \alpha_x@a \triangleright \_) = \Lambda$

No Adjoin (**vector definition**):

$$\frac{\langle \alpha_x@a \triangleright \boxed{\sigma}_y, i, j, k, l, \Lambda_1 \rangle}{\langle \alpha_x@a \triangleright \_, i, j, k, l, \Lambda \rangle}$$

$\mathsf{Valid}(\Lambda_1 \cup \{\boxed{\sigma} \mapsto \mathtt{na}\})$
$\mathsf{Filter}(\Lambda_1 \cup \{\boxed{\sigma} \mapsto \mathtt{na}\}, \alpha_x@a \triangleright \_) = \Lambda$

Figure 2.6: Modified inference rules for the CKY algorithm for TL-MCTAG. Alternative Adjoin, Substitute, and No Adjoin rules are given for the set and vector definitions. The item form, goal item and axioms are given in Figure 2.5.

will adjoin.

In order to directly parse tree-local MCTAG, items must keep track of the trees

set definition:
$\mathsf{Valid}(\Lambda)$ holds if for all links $\boxed{\sigma}_i$ and $\boxed{\sigma}_j$ in Λ, $\Lambda(\boxed{\sigma}_i) = \Gamma_x$ and $\Lambda(\boxed{\sigma}_j) = \Gamma_y$ and $x \neq y$ for some tree set Γ.

vector definition:
$\mathsf{Valid}(\Lambda)$ holds if for all links $\boxed{\sigma 1}$ and $\boxed{\sigma 2}$ in Λ, $\boxed{\sigma 1} \neq \boxed{\sigma 2}$.

Figure 2.7: Definition of the Valid condition, which ensures that all locations of a link are used by unique trees from the same tree set. Under the set definition there is an entry for each link location and both the identity of the tree set and the uniqueness of the tree from that tree set must be checked. Under the vector definition only the link name and the tree vector identifier are stored because the link locations uniquely select trees from within tree vectors.

that adjoin at each multicomponent link. We handle this by adding a **link history** to each item. Under the set definition, a link history is an associative array of links notated with indices and tree set identifiers notated with indices to identify a unique tree within the set. Note that because under the set definition a tree may adjoin at any location of a link, the indices of the link and tree set need not match. The axioms introduce empty link histories, indicating that no adjunctions have yet occurred. When an adjunction takes place, the tree identifier of the adjoining tree is associated with the link at which it adjoins. In order for an adjunction to take place at a multicomponent link, the adjoining tree's tree set must be the same as that of any tree identifier already stored for that link. This is enforced by the $\mathsf{Valid}(\Lambda)$ condition (Figure 2.7) defined on link histories. The $\mathsf{Filter}(\Lambda, \alpha@a \triangleright \ell)$ function removes links that are completely used from the argument link history. An empty link history indicates that tree locality has been enforced for the subtree specified by the item; thus no additional information need be maintained or passed on to later stages of the parse.

For the vector definition, the link histories may be simplified because each location

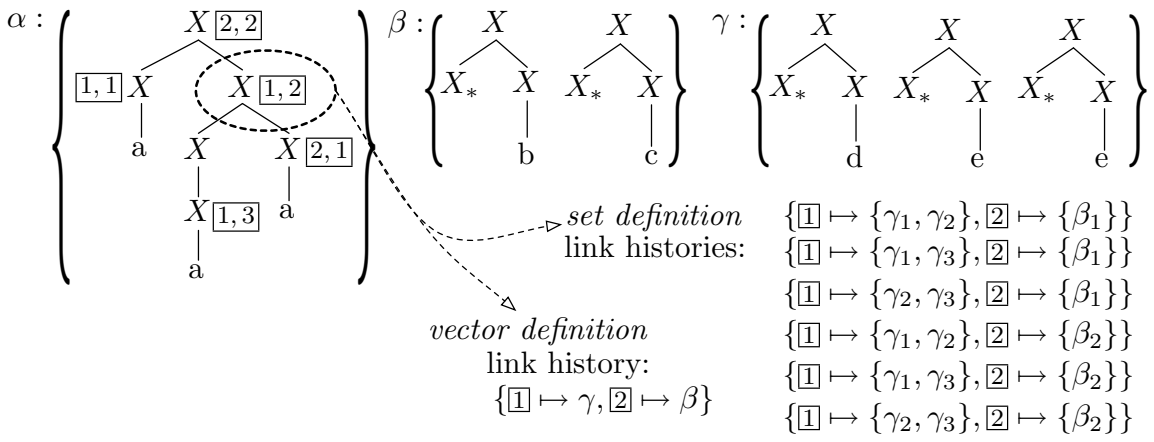


Figure 2.8: A sample TL-MCTAG with examples of the possible link histories under the set and vector definitions when the parser reaches the top of the circled node.

of a link fully specifies which tree from within a set may adjoin there. As a result, the link history is an associative array of links (not annotated with indices) and tree vector identifiers. An example contrasting the link histories for the set and vector definitions is given in Figure 2.8.

The addition of a link history to each item increases the complexity of the algorithm. The maximum link history length is bounded by the rank of the input grammar, r. Under the set definition, the number of possible values for each element of a link history is on the order of the number of tree sets in the grammar multiplied by the power set of the fan-out: $|G| \cdot 2^f$. Thus, for the set definition, the complexity of the algorithm is $O(n^6 \, |G|^{r+2} \, 2^{rf})$. Under the vector definition, the number of possible values for each element of a link history is on the order of the number of tree sets in the grammar. Thus, for the vector definition, the complexity of the algorithm is $O(n^6 \, |G|^{r+2})$. Note that the variable representing fan-out, f, is present only in the

complexity of the set definition. This demonstrates that when rank is bounded, even with unbounded fan-out, parsing the vector definition of TL-MCTAG is polynomial.

Permitting multiple adjunction may be accomplished by a method similar to the one described for the TAG algorithm. Rather than associating each node with at most one link, we permit nodes to be accompanied by a set of links. In contrast to the TAG case, here we must use a set rather than a list in order to allow the expressivity that multiple adjunction can provide. In the TAG case a list is sufficient because the links at a node are fully interchangeable. In the TL-MCTAG case, because the links are defined not just by the node where they appear but by the full set of nodes at which locations of that link appear, the links at a given node are not interchangeable. It must be possible to use them in any order.[7] Because the links can be used in any order, the addition of multiple adjunction adds a factor of 2^r to the time complexity of the parsing algorithm.

2.4 Link Factorization

The parser presented in the previous section has the advantage of running in polynomial time if the elementary structures of the input TL-MCTAG are defined as vectors and if the rank of the grammar is bounded by some constant. Bounding the rank by a constant might be too strong a limitation in natural language parsing applications, however. Thus, in the general case the running time of our algorithm contains a factor that is an exponential function of the rank of the input grammar.

[7]For links that share all locations it is still possible to enforce a strict order over them without compromising expressivity.

To optimize parsing time, then, we seek a method to "factorize" the elementary trees of the grammar in such a way that the rank is effectively reduced and the set of derived trees is preserved. In this section we present a novel and efficient algorithm for factorizing a TL-MCTAG into a strongly equivalent TL-MCTAG in which rank is minimized across the grammar. Here, strongly equivalent means that the two grammars generate the same set of derived trees with the exception of permitting a small, reversible transformation that is necessary to ensure that the factorized trees obey the TAG constraint that auxiliary trees must have matching root and foot node labels.

2.4.1 Preliminaries

Let α be some elementary tree. We write $|\alpha|$ to denote the number of nodes of α. For a link l, we write $|l|$ to denote the number of nodes of l.

For an elementary tree α, we call a **fragment** of α a complete subtree rooted at some node n of α, written $\alpha(n)$, or else a subtree rooted at n with a gap at node n' in its yield, written $\alpha(n, n')$. See Figure 2.9 for an example. We also use φ to denote a generic fragment with or without a gap node in its yield.

Consider some fragment φ of α. Let N_α be the set of all nodes of α and let N_φ be the set of nodes of φ with the exclusion of the gap node, in case φ has such a node. We say that φ is an **isolated** fragment if φ includes at least one link and no link in α impinges both on nodes in N_φ and on nodes in $N_\alpha - N_\varphi$. See Figure 2.9 for an example.

Intuitively, we can "excise" an isolated fragment from α without splitting apart

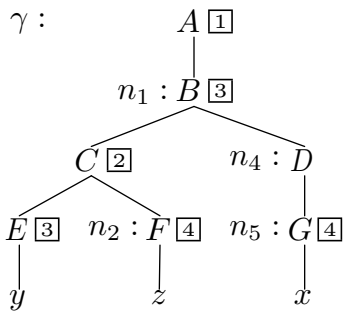


Figure 2.9: An elementary tree demonstrating fragments, isolation, and maximal nodes. Fragment $\varphi_1 = \alpha(n_1, n_2)$ contains all locations of links 2 and 3, because links at the root node of a fragment are contained within that fragment. It does not contain any locations of link 4, because links at the gap node of a fragment are not contained within that fragment. Because links 2 and 3 impinge only on nodes in φ_1 and all other links impinge only on nodes not in φ_1, φ_1 is an isolated fragment. Fragment $\varphi_2 = \alpha(n_4)$ is not an isolated fragment because it contains only one of the link locations of 4. Note also that n_4 is a maximal node but n_5 is not.

the links of α itself, and therefore preserving the tree locality. This operation may also reduce the number of links in α, which is our main goal. The factorization algorithm we present in Subsection 2.4.2 is based on the detection and factorization of isolated fragments.

Let n be a node from some elementary tree α. We write $\mathsf{lnodes}(n)$ to denote the set of all nodes from fragment $\alpha(n)$ that are part of some link from α. Node n is **maximal** if

- $\mathsf{lnodes}(n) \neq \emptyset$; *and*
- n is either the root node of α or, for its parent node n', we have $\mathsf{lnodes}(n') \neq \mathsf{lnodes}(n)$.

Note that for every node n' of α such that $\mathsf{lnodes}(n') \neq \emptyset$ there is always a unique maximal node n such that $\mathsf{lnodes}(n') = \mathsf{lnodes}(n)$. See Figure 2.9 for an example.

Thus, for the purpose of TL-MCTAG factorization, we can consider only maximal nodes. The first item in the definition of maximal node, stating that a maximal node always dominates (possibly reflexively) some node involved in a link, will often be implicitly used below.

We need to distinguish the nodes in $\mathsf{lnodes}(n)$ depending on their impinging links. Assume that $\{l_1, l_2, \ldots, l_r\}$ is the set of all links occurring in α. For $1 \leq j \leq r$, we write $\mathsf{lnodes}(n, l_j)$ to denote the set of all nodes from fragment $\alpha(n)$ with impinging link l_j. Thus, $\bigcup_{j=1}^{r} \mathsf{lnodes}(n, l_j) = \mathsf{lnodes}(n)$. We associate with each maximal node n of α a **signature** $\sigma(n)$, defined as an integer vector of size r. For each j, $1 \leq j \leq r$, we define

$$\sigma(n)[j] \;=\; \begin{cases} \mathsf{lnodes}(n, l_j), & \text{if } 0 < |\mathsf{lnodes}(n, l_j)| < |l_j|\,; \\ \emptyset, & \text{if } |\mathsf{lnodes}(n, l_j)| = 0 \text{ or} \\ & |\mathsf{lnodes}(n, l_j)| = |l_j|\,. \end{cases}$$

Observe that, in the above definition, $\sigma(n)[j] = \emptyset$ means that none or all of the nodes of l_j are found within fragment $\alpha(n)$. The **empty** signature, written $\mathbf{0}$, is the signature with all of its components set to $\emptyset$.

Consider maximal nodes n_1 and n_2 such that $n_1 \neq n_2$, $\sigma(n_1) \neq \mathbf{0}$ and $\sigma(n_2) \neq \mathbf{0}$. It is not difficult to see that $\sigma(n_1) = \sigma(n_2)$ always implies that one of the two nodes dominates the other. This observation is implicitly used in several places below.

When visiting nodes of α in a path from some leaf node to the root node,[8] one may encounter several maximal nodes having the same non-empty signature. In our

[8]We view trees as directed graphs with arcs directed from each node to its parent.

factorization algorithm, we need to consider pairs of such nodes that are as close as possible. Consider two maximal nodes n_1 and n_2, $n_1 \neq n_2$, such that n_1 dominates n_2. The ordered pair (n_1, n_2) is called a **minimal pair** if $\sigma(n_1) = \sigma(n_2) \neq \mathbf{0}$ and, for every maximal node n_3 in the path from n_2 to n_1 with $n_3 \neq n_1$ and $n_3 \neq n_2$, we have $\sigma(n_3) \neq \sigma(n_1)$. Consider now a sequence $\langle n_1, n_2, \ldots, n_q \rangle$, $q \geq 2$, of nodes from α. Such a sequence is called a **maximal chain** if each pair (n_{i-1}, n_i) is a minimal pair, $2 \leq i \leq q$, and all nodes n from α with $\sigma(n) = \sigma(n_1)$ are included in the sequence itself.

Notice that two maximal nodes belonging to two different maximal chains must have different signatures, and thus one maximal node cannot belong to more than one maximal chain. We now prove some basic properties of the notions introduced above, that will be used later in the development of our factorization algorithm and in the proof of some of its mathematical properties.

Lemma 1

Let α be an elementary tree and let n, n' be maximal nodes, with n properly dominating n' in (ii) below.

(i) $\sigma(n) = \mathbf{0}$ if and only if $\alpha(n)$ is an isolated fragment;

(ii) $\sigma(n) = \sigma(n')$ if and only if $\alpha(n, n')$ is an isolated fragment.

Proof

(i). If $\sigma(n) = \mathbf{0}$, then for each link l we have that either all nodes impinged on by l are dominated (possibly reflexively) by n or none of these nodes is dominated by n. Since n is maximal, we further conclude that at least some link l is found within

$\alpha(n)$.

Conversely, if $\alpha(n)$ is an isolated fragment then all or none of the nodes impinged on by some link l are dominated by n, and thus $\sigma(n) = \mathbf{0}$.

(ii). Let $\sigma(n) = \sigma(n')$, with n properly dominating n'. For each link l_j, there are two possible cases. First consider the case where $\sigma(n)[j] = \sigma(n')[j] = \emptyset$. In order for this to be true, the link must be in one of three configurations, all of which satisfy the requirement that the locations of l_j must be all inside or all outside of the fragment $\alpha(n_1, n_2)$.

- $\mathsf{Inodes}(n, j) = \emptyset$. In this configuration no one of the nodes on which l_j impinges is dominated by n.
- $|\mathsf{Inodes}(n, j)| = |l_j|$. We distinguish two possible cases.
  - $\mathsf{Inodes}(n', j) = \emptyset$. In this configuration all the nodes on which l_j impinges are within the fragment $\alpha(n_1, n_2)$.
  - $|\mathsf{Inodes}(n', j)| = |l_j|$. In this configuration all the nodes on which l_j impinges are "below" the fragment $\alpha(n, n')$.

Now consider the case where $\sigma(n)[j] = \sigma(n')[j] \neq \emptyset$. The nodes in $\mathsf{Inodes}(n', j)$ are dominated (possibly reflexively) by n' and therefore fall "below" $\alpha(n, n')$. The remaining nodes on which l_j impinges cannot be dominated (possibly reflexively) by n. We thus conclude that no nodes impinged on by l_j occur within the fragment $\alpha(n, n')$.

Assume now that $\alpha(n, n')$ can be isolated. We can use exactly the same arguments as above in the analysis of sets $\mathsf{Inodes}(n, j)$ and $\mathsf{Inodes}(n', j)$, and conclude that $\sigma(n) =$

$\sigma(n')$. ■

The next lemma will be useful later in establishing that the factorization found by our algorithm is optimal.

Lemma 2

Let (n_1, n_2) be some minimal pair. Then

(i) for any node n_3 in the path from n_2 to n_1, $\sigma(n_3) \neq \mathbf{0}$;

(ii) for any minimal pair (n_3, n_4), neither or both of n_3 and n_4 are found in the path from n_2 to n_1.

Proof

(i). Because $\sigma(n_2) \neq \mathbf{0}$, there is some link l_j for which $\sigma(n_2)[j] = \mathsf{Inodes}(n_2, j) \neq \emptyset$. Because n_3 dominates n_2, n_3 dominates the nodes in $\mathsf{Inodes}(n_2, j)$. Therefore, the only way $\sigma(n_3)$ could equal $\mathbf{0}$ is if $|\mathsf{Inodes}(n_3, j)| = |l_j|$. But then $\sigma(n_1)[j] = \emptyset$ because n_1 dominates n_3. This is a contradiction.

(ii). Assume that n_4 is on the path from n_2 to n_1. From the definition of minimal pair, there must exist a link l_k such that $\sigma(n_4)[k] \neq \sigma(n_2)[k]$. By the same reasoning as in the proof of statement (i) above, for any link l_j such that $\sigma(n_2)[j] \neq \emptyset$, we must have $\sigma(n_2)[j] = \sigma(n_4)[j] = \sigma(n_1)[j]$. We thus conclude that $\sigma(n_2)[k] = \emptyset$ and $\sigma(n_4)[k] \neq \emptyset$. Since $\sigma(n_4)[k] = \sigma(n_3)[k] \neq \emptyset$ and $\sigma(n_2)[k] = \sigma(n_1)[k] = \emptyset$, node n_3 must be in the path from n_2 to n_1.

By a similar argument, we can argue that if n_3 is on the path from n_2 to n_1, then node n_4 must be in that path as well. ■

1: **Function** CHAIN(α) {α an elementary tree from a TL-MCTAG}
2: $\mathcal{L} \leftarrow \emptyset$; {associative array mapping signatures into node lists}
3: **for all** maximal nodes n from α, in top down order **do**
4: **if** $\sigma(n) \neq \mathbf{0}$ **then**
5: append n to list $\mathcal{L}(\sigma(n))$;
6: mark as maximal chain each list in $\mathcal{L}$

Figure 2.10: Construction of maximal chains in the factorization algorithm.

2.4.2 Factorization algorithm

Let G be an input TL-MCTAG grammar. In this subsection we provide a method for the construction of a TL-MCTAG that is strongly equivalent in generative power to G and that has minimal rank. We start with the discussion of some preprocessing of the input.

We annotate each elementary tree α as explained in what follows: We compute sets $\mathsf{lnodes}(n, l_j)$ for all nodes n and all links l_j of α. This can easily be done with a bottom up visit of α, by observing that if an internal node n has children $n_1, n_2, \ldots, n_k$ then $\mathsf{lnodes}(n, l_j) = \bigcup_{i=1}^{k} \mathsf{lnodes}(n_i, l_j)$. Using sets $\mathsf{lnodes}(n, l_j)$, we can then mark all nodes n in α that are maximal, and compute the associated signatures $\sigma(n)$.

We also mark all maximal chains within α. This simple procedure is reported in Figure 2.10. We maintain an associative array with node signatures as entries and node lists as values. We visit all maximal nodes of α in a top down fashion, creating a list for each different signature and appending to such a list all nodes having that signature.

In the algorithm below we excise isolated fragments from each elementary tree α. We now introduce some conventions for doing this. Although it would be possible to excise fragments without the introduction of additional tree structure, we adopt

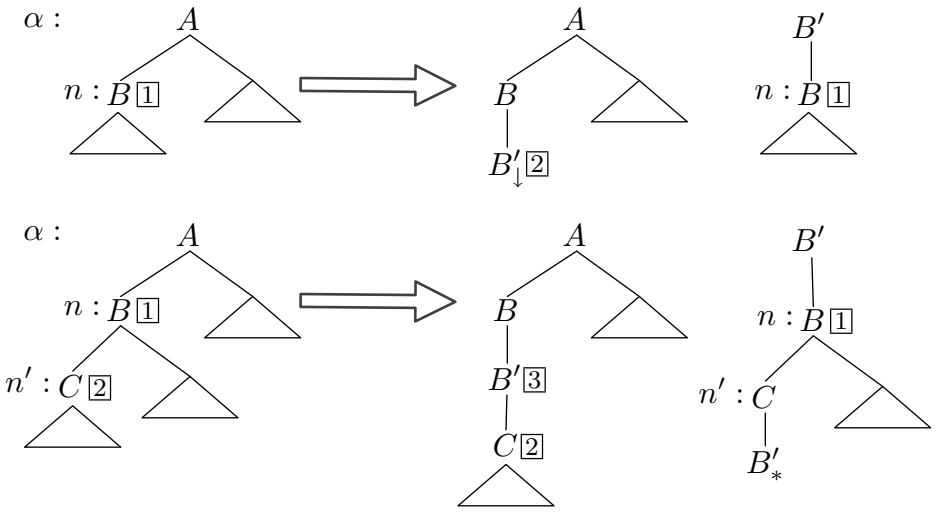


Figure 2.11: Diagrams of the tree transformations performed when fragments $\alpha(n)$ and $\alpha(n, n')$ are removed.

instead two simple tree transformations that preserve auxiliary tree root and foot label matching and result in some simplification of the notation used by the algorithm, particularly in case the root node of a fragment is the same as the gap node of a second fragment within α. A schematic depiction of both transformations is given in Figure 2.11.

When a fragment $\alpha(n)$ is excised, we leave a copy of the root node n without its impinging links that dominates a fresh node n' with a fresh link indicating obligatory substitution of the excised fragment. The excised fragment consists of $\alpha(n)$ including any links impinging on n, but has a fresh root node immediately dominating n with the same label as n'. After substitution of the excised fragment $\alpha(n)$ back into node n', the original tree can be easily regenerated by removing the fresh node n' and by merging the two copies of node n.

A similar transformation is used to excise a fragment $\alpha(n, n')$ that does not collapse

the root node n and the gap node n'. Nodes n and n' of the original tree are not altered, and thus they retain their names. The material between them is replaced with a single new node with a fresh nonterminal symbol and a fresh link. This link indicates the obligatory adjunction for a transformation of the excised fragment. The transformation adds to $\alpha(n, n')$ a new root node and a new gap node, both with the same label, in order to make it technically possible to perform adjunction. We remark that any link impinging on the root node of the excised fragment is by our convention included in the excised fragment, and any link impinging on the gap node is not.

To regenerate the original tree, the excised fragment $\alpha(n, n')$ can be adjoined back into the tree from which it was excised. The new nodes that have been generated in the excision may be removed and the original root and gap nodes may be merged back together retaining any impinging links, respectively. Note that if there was a link on either the root or gap node in the original tree, it is not lost or duplicated in the process.

We need to introduce one more convention for tree excision. Consider a maximal chain $c = \langle n_1, n_2, \ldots, n_q \rangle$ in α, $q \geq 2$. In case $q = 2$, our algorithm processes c by excising a fragment $\alpha(n_1, n_2)$ from α, exactly as explained above. In case $q > 2$, a special processing is required for c. Chain c represents $q - 1$ minimal pairs, corresponding to fragments $\alpha(n_{i-1}, n_i)$, $2 \leq i \leq q$. We do not excise these $q - 1$ fragments one by one, because this would create $q - 1 > 1$ new links within α. We follow instead a procedure that "binarizes" c, as explained below.

Let us recursively define elementary tree α_c as follows.

- In case $q = 3$, α_c is a tree composed of two nodes n and n', with n immedi-

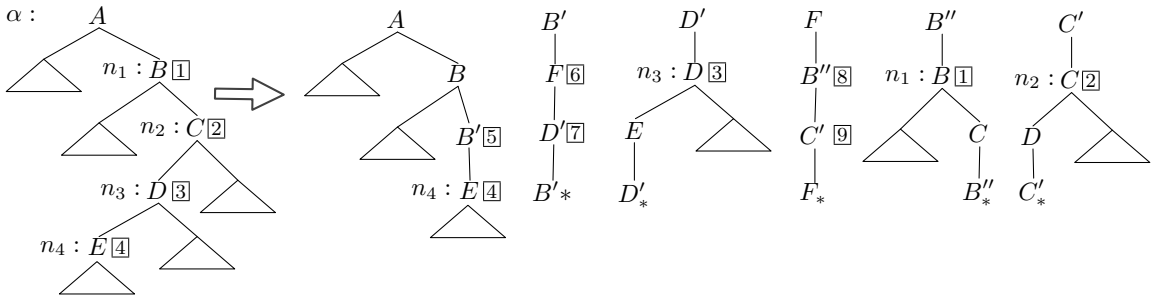


Figure 2.12: The binarization procedure applied to maximal chain $c = \langle n_1, n_2, n_3, n_4 \rangle$.

ately dominating n'. Node n hosts the (obligatory) adjunction of the fragment $\alpha(n_1, n_2)$ and node n' hosts the (obligatory) adjunction of $\alpha(n_2, n_3)$. Both fragments are transformed as previously discussed.

- In case $q > 3$, α_c is a two node tree specified as above, with n' hosting the (obligatory) adjunction of the transformed fragment $\alpha(n_{q-1}, n_q)$. Node n hosts the adjunction of tree $\alpha_{c'}$, with $c' = \langle n_1, \ldots, n_{q-1} \rangle$.

Note that each tree α_c has rank two.

When processing a maximal chain c with $q > 2$, the whole fragment $\alpha(n_1, n_q)$ is excised, using the convention above. This results in a single fresh link added to α. In this case the link refers to the adjunction of a newly created elementary tree α_c, defined as above. An example of the binarization of a maximal chain with $q = 4$ is reported in Figure 2.12.

We can now discuss the factorization algorithm, reported in Figure 2.13. For a maximal node n in an elementary tree α, we write $\mathsf{links}(n)$ to denote the number of links from α that are entirely contained in fragment $\alpha(n)$. We process each tree set Γ of the source grammar and each elementary tree α in Γ as follows.

```
1: Function FACTORIZE(G) {G a tree-local MCTAG}
2: G' ← tree-local MCTAG with no tree sets;
3: for all tree sets Γ from G do
4:   for all elementary trees α in Γ do
5:     B ← ∅; {priority queue used as an agenda}
6:     for all maximal nodes n from α other than the root do
7:       if σ(n) = 0 then
8:         add n to B with score links(n);
9:     for all maximal chains ⟨n_1, ..., n_q⟩ from α do
10:      add ⟨n_1, ..., n_q⟩ to B with score links(n_1) − links(n_q);
11:    while B ≠ ∅ do
12:      pop from B item I with smallest score, discarding items with score = 1;
13:      if I = n then
14:        α ← excise α(n) from α;
15:        add to G' tree set {α(n)};
16:      if I = ⟨n_1, n_2⟩ then
17:        α ← excise α(n_1, n_2) from α;
18:        add to G' tree set {α(n_1, n_2)};
19:      if I = ⟨n_1, ..., n_q⟩, q > 2 then
20:        α ← excise α(n_1, n_q) from α;
21:        for all i with 2 ≤ i ≤ q do
22:          add to G' tree set {α(n_{i−1}, n_i)};
23:          add to G' tree set {α_c} with c = ⟨n_1, ..., n_i⟩;
24:  add tree set Γ to G'
25: return G'
```

Figure 2.13: The factorization algorithm for tree-local MCTAG.

On a first phase, we add to an agenda B each maximal node n different from the root of α and such that $\sigma(n) = \mathbf{0}$. We associate this agenda item with the score $\mathsf{links}(n)$. At the same time, each maximal chain $\langle n_1, n_2, \ldots, n_q \rangle$, $q \geq 2$, is added to B, with associated score $\mathsf{links}(n_1) - \mathsf{links}(n_q)$.

On a second phase, we process all items in B, in order of increasing score and ignoring those items that have score of one. If the current item is a maximal node n, we excise the fragment $\alpha(n)$ from α, leaving in place a fresh node with a single node link denoting obligatory substitution. If the current item is a maximal chain of

the form $\langle n_1, n_2 \rangle$, we excise from α the fragment $\alpha(n_1, n_2)$, leaving in place a fresh node with a single node link denoting obligatory adjunction of the excised fragment. Finally, if the current item is a maximal chain $c = \langle n_1, \ldots, n_q \rangle$ with $q > 2$, we excise from α the whole fragment $\alpha(n_1, n_q)$, and we apply to the chain the binarization procedure described in this subsection. This results in the addition to the output grammar of fragments $\alpha(n_{i-1}, n_i)$, for $2 \leq i \leq q$, and of newly created elementary tree α_c and elementary trees $\alpha_{c'}$ for each chain c' that is a proper prefix of c. After the processing of all elementary trees in tree set Γ is completed, the resulting version of set Γ is also added to the output grammar.

The discussion of the correctness of the algorithm is reported in the next subsection, along with some other mathematical properties.

2.4.3 Mathematical properties

We discuss in this subsection some mathematical properties of our factorization algorithm. Let G be the input TL-MCTAG and let G' be the output of the algorithm. We start with the issue of correctness. First, notice that our algorithm stops after a finite number of steps, since the number of possible excisions for G is finite. Assume now that φ and φ' are two isolated fragments within some elementary tree α, and φ' is itself a fragment within φ. It is easy to see that excising φ' from φ results in a new fragment of α that is still an isolated fragment. Using this observation together with Lemma 1, we can then conclude that all fragments that are excised by the algorithm are isolated fragments. This in turn implies that each fragment excision in our algorithm preserves tree locality, and G' is still a TL-MCTAG.

Each fragment that is excised from some source tree must obligatorily be adjoined back into that tree, at the point from which it was removed. Thus, G' is generatively equivalent in the strong sense to G, modulo our trivial tree transformation for the root and the gap nodes. This proves the correctness of our factorization algorithm.

One remark is in order here. Note that we always excise fragments that have at least two links. This can be shown inductively as follows. Consider first the smallest fragments that are excised from some elementary tree α, that is, those fragments that do not contain any other fragment within themselves. These fragments always have at least two links, because of the requirement stated in line 12 in the algorithm. In the inductive case, let φ be some fragment of α from which a second fragment φ' has been already excised in some iteration of the loop at lines from 11 to 23. Fragment φ' is thus replaced by some link l'. Because of the definition of maximal node, φ must contain at least one link l that is not contained in φ'. In case l itself is part of some excised fragment φ'', there will still be some other fresh link replacing φ''. We thus conclude that, when excised, φ always has at least two links. Since excised fragments always have at least two links and since we never consider elementary trees as candidate fragments (line 6), we can conclude that our algorithm always finds a non-trivial factorization of G.

We can now turn to an analysis of the computational complexity of our algorithm. Consider an elementary tree α of G with r links and with a maximum of f nodes per link. In the preprocessing phase of the algorithm, the computation of sets $\mathsf{lnodes}(n, l_j)$ can be carried out in time $\mathcal{O}(|\alpha| \cdot r \cdot f)$. To see this, notice that there are no more than $|\alpha| \cdot r$ sets $\mathsf{lnodes}(n, l_j)$. Furthermore, we have $|\mathsf{lnodes}(n, l_j)| \leq f$, and each

node in $\mathsf{lnodes}(n, l_j)$ is processed in constant time through the union operator, when constructing set $\mathsf{lnodes}(n', l_j)$ for the parent node n' of n. Clearly, $\mathcal{O}(|\alpha| \cdot r \cdot f)$ is also a time upper bound for the computation of quantities $\sigma(n)$ and $\mathsf{links}(n)$ for all nodes in α, and for extracting a list of the maximal nodes therein as well.[9]

In what follows, we will need to compare signatures of different nodes for equality. Despite the fact that each signature has r elements, and each element of a signature is a set with $\mathcal{O}(|f|)$ elements, there are at most $|\alpha|$ different signatures. We can therefore use an atomic symbol to name each signature (perfect hashing). In this way, signatures can be compared in constant time.

The marking of all maximal chains within α, as specified by the algorithm in Figure 2.10, can be implemented in time $\mathcal{O}(|\alpha|)$. This is done by encoding the associative array $\mathcal{L}$ in the algorithm through a one-dimensional array indexed by signature names. Each element of the array points to a linked list of nodes, representing a maximal chain.

We now analyze the running time of the factorization function in Figure 2.13. Let us first consider a single elementary tree α. We implement the priority queue B through a heap data structure. The loops at lines 6 and 9 run in time $\mathcal{O}(|\alpha| \cdot \log(|\alpha|))$: this is the standard result for populating a heap; see for instance Cormen et al. [2001]. At each iteration of the while loop at lines 11 to 23, we extract some fragment $\alpha(n)$ or $\alpha(n_1, n_q)$. The processing of each such fragment φ takes an amount of time $\mathcal{O}(|\varphi|)$, where $|\varphi|$ is the number of nodes of φ. In such an iteration, α needs to be re-edited into

[9]We remark here that a further improvement in efficiency could be achieved by replacing the sets of nodes in a signature with the single node that is the least common ancestor of the set of nodes. However, using the set of nodes substantially improves the clarity of the presentation of the algorithm, so we do not pursue this optimization here.

a new elementary tree with a number of nodes $|\alpha| - |\varphi| + c$, where $c \leq 3$ is a constant that depends on the specific transformation in Figure 2.11 that was applied in the excision of the fragment tree. Nonetheless, if a suitable representation is maintained for α, making use of nodes and pointers, the re-editing of α can be done in constant time. Then a single iteration of the while loop takes time $\mathcal{O}(|\varphi|)$, with φ the excised fragment. We can then conclude that all iterations of the while loop take an amount of time $\mathcal{O}(|\alpha| \cdot \log(|\alpha|))$, where the $\log(|\alpha|)$ factor accounts for the management of the heap; see again Cormen et al. [2001].

Now let α_M be the elementary tree of G with largest size, and let r_G and f_G be the rank and fan-out of G, respectively. Putting everything together, the total running time of the factorization algorithm is a function in $\mathcal{O}(|G| \cdot (r_G \cdot f_G + \log(|\alpha_M|)))$, where $|G|$, the size of the input grammar, is defined as the sum of terms $|\alpha|$ for all elementary trees α of G. Since we always have $f_G \leq |\alpha_M|$, the previous upper bound can be rewritten as $\mathcal{O}(|G| \cdot |\alpha_M| \cdot r_G)$.

A special case is worth discussing here. If the maximum number of links impinging on a node of our elementary trees is bounded by some constant, we have $r_G \cdot f_G = \mathcal{O}(|\alpha_M|)$. In this case, the above bound reduces to $\mathcal{O}(|G| \cdot |\alpha_M|)$. The constant bound on the number of links impinging on the nodes of a grammar holds for all of the grammars we have exploited in Section 2.1.

We now argue that our algorithm provides the factorization G' of G with the smallest possible rank, under the assumption that G and G' are strongly equivalent, that is, they generate the same derived trees.

A factorization f of G is called **maximal** if no one of its fragments has a smaller

isolated fragment within itself. We start by observing that the factorization of G found by our algorithm is maximal. To see this, consider the excision by our algorithm of a maximal chain $\langle n_1, \ldots, n_q \rangle$ within an elementary tree α. This item is added to the priority heap at line 10, with a score of $\mathsf{links}(n_1) - \mathsf{links}(n_q)$. The score is the number of links found in fragment $\alpha(n_1, n_q)$, with the exclusion of the links at the gap node n_q. The chain is then factorized into fragments $\alpha(n_{i-1}, n_i)$, for each i with $2 \leq i \leq q$. Assume that some fragment $\alpha(n_{i-1}, n_i)$ contains in turn a maximal chain $\langle n'_1, \ldots, n'_{q'} \rangle$ or else an isolated fragment of the form $\alpha(n')$. In the first case we have $\mathsf{links}(n'_1) - \mathsf{links}(n'_{q'}) < \mathsf{links}(n_1) - \mathsf{links}(n_q)$ and in the second case we have $\mathsf{links}(n') < \mathsf{links}(n_1) - \mathsf{links}(n_q)$. Thus the smaller chain or fragment is processed earlier than our maximal chain, and by the time our maximal chain is processed, the smaller chain or fragment has already been excised. A similar argument shows that the excision by our algorithm of an isolated fragment of the form $\alpha(n)$ happens after the excision of any maximal chain or fragment included within $\alpha(n)$ itself.

We now show that the maximal factorization of G is unique. Let φ and φ' be two isolated fragments of some elementary tree α. We say that φ and φ' **partially overlap** if the set of nodes shared by φ and φ' is not empty and is a proper subset of the nodes of both fragments. It is not difficult to see that if φ and φ' partially overlap, then at least one among φ and φ' must have the form $\alpha(n_1, n_2)$.

Without any loss of generality, we assume that the elementary trees of G are always factorized at their maximal nodes, as discussed in Subsection 2.4.1. Let us assume that f and f' are two distinguishable maximal factorizations of G. Since no fragment of one factorization can be a sub-fragment of some fragment of the other

factorization, there must be some fragment φ of f and some fragment φ' of f' such that f and f' partially overlap.

Assume that φ has the form $\alpha(n_1)$. Then φ' must have the form $\alpha(n_2, n_3)$, and n_1 must be in the path from n_3 to n_2. Since φ' is as small as possible, (n_2, n_3) must be a minimal pair. We have then established a violation of Lemma 2(i). Assume now that φ has the form $\alpha(n_1, n_2)$. Again, (n_1, n_2) must be a minimal pair. If φ' has the form $\alpha(n_3)$, the above argument applies again, resulting in a violation of Lemma 2(i). If φ' has the form $\alpha(n_3, n_4)$, then (n_3, n_4) must be a minimal pair. Furthermore, n_1, n_2, n_3 and n_4 must all be on the same path within α, with n_1, n_2 in alternation with n_3, n_4. This establishes a violation of Lemma 2(ii). The assumption that f and f' partially overlap then leads to a contradiction, and we must conclude that the maximal factorization of G is unique.

We can also use the above argument against the existence of overlapping fragments to show that any factorization f of G other than the unique maximal factorization f_M must be coarser than f_M, meaning that each fragment φ of f is a also a fragment of f_M, or else φ can be represented as a combination of the fragments of f_M (through substitution and adjunction). This means that no factorization of G can have rank smaller than the rank of the maximal factorization f_M. We conclude that our algorithm is optimal.

The above discussion on the optimality of the factorization algorithm crucially assumes strong equivalence with the source TL-MCTAG G. Of course there might be TL-MCTAGs that are weakly equivalent to G, that is, they generate the same language, and have rank strictly smaller than the rank of G'. However, finding such

structurally different grammars is a task that seems to require techniques quite different from the factorization techniques we have developed in this section. Furthermore, the task might be computationally unfeasible, considering the fact that the weak equivalence problem for TL-MCTAG is undecidable. (Such a problem is undecidable even for context-free grammars).

We remark here that if we are allowed to change G by recasting its elementary trees in some suitable way, we might still be able to further reduce the rank with respect to the algorithm we have presented in this section. In this case the output grammar would not preserve the derived trees, that is, we lose the strong equivalence, but still retain the derivation trees unaltered. One such case arises when the input TL-MCTAG is not in binary form, that is, some nodes have more than two children. If we allow binarization of the elementary trees of the source grammar, then we might be able to isolate sets of links that could not be factorized in the source grammar itself. Currently, the definition of fragment does not allow splitting apart a subset of the children of a given node from the remaining ones. However, the number of binarizations of an elementary tree is not bounded by a polynomial function of the size of the tree itself, and choosing the binarization that leads to optimal rank reduction is a challenging problem. This new optimization task has the flavor of other unsupervised learning problems from the literature on machine learning and grammar induction that are computationally hard. We leave this problem for future work.

A second case arises when multiple links impinge on the same node of an elementary tree. As presented, the factorization algorithm is designed to handle grammars in which multiple adjunction is permitted. However, if multiple adjunction is disal-

lowed and the grammar contains trees in which multiple links impinge on the same node, the use of one link at a node will disqualify any other impinging links from use. This opens up the possibility of further reducing the rank of the grammar by producing tree sets that do not contain any nodes on which multiple links impinge. This can be accomplished by performing a first pass grammar transformation in which a copy of each elementary tree set is added to the grammar for each distinct, maximal, non-conflicting set of links appearing in the tree set. This transformation in itself may result in a reduction of the rank of the source grammar. The factorization algorithm can then be applied on the new grammar. However, if the elementary trees in the source grammar contain clusters of links that are mutually overlapping, the suggested transformation may blow up the size of the input grammar in a way that is not bounded by any polynomial function.

2.5 Conclusion

This chapter explores the complexity of TL-MCTAG, showing that recognition is NP-complete under a range of interesting restrictions. It then provides a parsing algorithm that performs better than the extrapolation of the standard multiple CFG parsing method to TL-MCTAG. As shown by our proofs, the difficulty in parsing TL-MCTAG stems from the rank of the input grammar. We offer a novel and efficient algorithm for minimizing the rank of the input grammar while preserving its strong generative capacity.

Our work on TL-MCTAG complexity bears comparison to that of several others. Kallmeyer [2007] provides a clear and insightful breakdown of the different character-

istics of MCTAG-variant grammars and the effect of these characteristics on expressivity and complexity. That work clarifies the definitions of MCTAG variants and the relationship between them rather than presenting new complexity results. However, it suggests the possibility of proving results such as ours in its assertion that after a standard TAG parse a check of whether particular trees belong to the same tree set cannot be performed in polynomial time. Kallmeyer [2007] also addresses the problem of parsing MCTAG, although not specifically for TL-MCTAG. The method proposed differs from ours in that MCTAGs are parsed first as a standard TAG with any conditions on tree or set locality checked on the derivation forest as a second step. No specific algorithm is presented for performing the check of tree-locality on a TAG derivation forest, so it is difficult to directly compare the methods. However, that method cannot take advantage of the gains in efficiency produced by discarding inappropriate partial parses at the time that they are first considered. Aside from Kallmeyer's work, little attention has been paid to the problem of directly parsing TL-MCTAG.

Søgaard et al. [2007] present several proofs regarding the complexity of the recognition problem for some linguistically motivated extensions of TAG that are similar to TL-MCTAG. Their work shows NP-hardness of the recognition problem for these variants and, as an indirect result, also demonstrates the NP-hardness of TL-MCTAG recognition. This work differs from ours in that it does not show the NP-completeness of TL-MCTAG recognition and does not further locate and constrain the source of the NP-completeness of the problem to the rank of the input grammar nor does it provide mitigation through rank reduction of the grammar.

Our work on TL-MCTAG factorization is thematically though not formally related to the body of work on induction of TAG grammars from a treebank exemplified by Chen and Shanker [2004b]. The factorization performed in their work is done on the basis of syntactic constraints rather than with the goal of reducing complexity. Working from a treebank of actual natural language sentences, their work does not have the benefit of explicitly labeled adjunction sites but rather must attempt to reconstruct a derivation from complete derived trees.

The factorization problem we address is more closely related to work on factorizing synchronous CFG [Gildea et al., 2006, Zhang and Gildea, 2007] and on factorizing synchronous TAG [Nesson et al., 2008a]. Synchronous grammars are a special case of multicomponent grammars, so the problems are quite similar to the TL-MCTAG factorization problem. However, synchronous grammars are fundamentally set-local rather than tree-local formalisms, which in some cases simplifies their analysis. In the case of CFG, the problem reduces to one of identifying problematic permutations of non-terminals [Zhang and Gildea, 2007] and can be done efficiently by using a sorting algorithm to binarize any non-problematic permutations until only the intractable correspondences remain [Gildea et al., 2006]. This method is unavailable in the TAG case because the elementary structures may have depth greater than one and therefore the concept of adjacency relied upon in their work is inapplicable. The factorization algorithm of Nesson et al. [2008a] presented in this thesis in Chapter 4 is the most closely related to this one but is not directly applicable to TL-MCTAG because each link is presumed to have exactly two locations.

In conclusion, this work falls in an active line of research into efficient processing of

multicomponent and synchronous formalisms that appear computationally intractable but have desirable characteristics for meeting the expressive needs of natural language. It presents novel complexity results and algorithms for TL-MCTAG, a widely known and used formalism in computational linguistics that may be applied more effectively in natural-language processing with the introduction of algorithms to process it as efficiently as possible.

Chapter 3

Extensions to Tree-Local MCTAG

Although much work in TAG syntax and semantics makes use of TL-MCTAG to model phenomena such as quantifier scoping, Wh-question formation, and many other constructions [Kallmeyer and Romero, 2004, Romero et al., 2004], certain applications appear to require even more flexibility than is provided by TL-MCTAG. Scrambling is one well-known example [Rambow, 1994]. In addition, in the semantics domain, the use of a new TAG operation, flexible composition, is used to perform certain semantic operations that seemingly cannot be modeled with TL-MCTAG alone [Chiang and Scheffler, 2008] and in work in synchronous TAG semantics, constructions such as nested quantifiers require a set-local analysis [Nesson and Shieber, 2006].

One potential solution is to extend the domain of locality beyond that of TL-MCTAG. SL-MCTAG is the obvious choice for an extension, but, as was discussed in the previous chapter, it is substantially more difficult to parse even than TL-MCTAG. In addition, it is not clear that the extension is sufficient to handle the linguistic challenges posed by the abovementioned problems.

In this chapter we suggest addressing this problem with a shift in focus from constraining locality and complexity through restrictions that all trees in a tree set must adjoin within a single tree or tree set to constraining locality and complexity through restrictions on the derivational distance between trees in the same tree set in a valid derivation. We examine three formalisms, two of them introduced here for the first time, that use derivational distance to constrain locality and demonstrate by construction of parsers their relationship to TL-MCTAG in both expressivity and complexity.

In Section 3.1 we elaborate further the distinction between these two types of locality restrictions using TAG derivation trees. In Section 3.2 we address the simultaneity condition present in all MCTAGs and discuss the consequences of dropping it. In Sections 3.3 and 3.4 we introduce two novel formalisms, restricted non-simultaneous MCTAG and restricted Vector-TAG, respectively, and define a CKY-style parsers for them. We also introduce a variant of restricted Vector TAG, Limited Delay Vector-TAG, which we make use of as the base formalism for our work in Chapter 6. In Section 3.5 we recall the delayed TL-MCTAG formalism introduced by Chiang and Scheffler [2008] and define a CKY-style parser for it as well. In Section 3.6 we explore the complexity of all three parsers, the relationship between the formalisms.

3.1 Domains of Locality and Derivation Trees

The domains of locality of TL-MCTAG and SL-MCTAG (and trivially, TAG) can be thought of as lexically defined. That is, all locations at which the adjunction of one tree set into another may occur must be present within a single lexical item.

However, we can also think of locality derivationally. In a derivationally local system the constraint is on the relationships allowed between members of the same tree set in the derivation tree.

TAG derivation trees provide the information about how the elementary structures of the grammar combine that is necessary to construct the derived tree. Nodes in a TAG derivation tree are labeled with identifiers of elementary structures. One elementary structure is the child of another in the derivation tree if it adjoins or substitutes into it in the derivation. Arcs in the derivation tree are labeled with the address in the target elementary structure at which the operation takes place.

In MCTAG the derivation trees are often drawn with identifiers of entire tree sets as the nodes of the tree because the lexical locality constraints require that each elementary tree set be the derivational child of only one other tree set. This method of drawing derivation trees obfuscates a stark contrast in the derivational locality of these two formalisms. As an alternative, we use an **elaborated derivation tree** in which each tree from a tree set appears as a node in a tree. In TL-MCTAG all trees in a set must adjoin to the same tree. This means that they must all be siblings in the elaborated derivation tree. In SL-MCTAG, on the other hand, it is possible to generate derivations with arbitrarily long distances before the nearest common ancestor of two trees from the same elementary tree set is reached. An example SL-MCTAG grammar that can produce an arbitrarily long derivational distance to the nearest common ancestor of the trees in a given tree set is given in Figure 3.1 along with an example standard and elaborated derivation tree.

Chiang and Scheffler [2008] recently introduced one variant of MCTAG, delayed

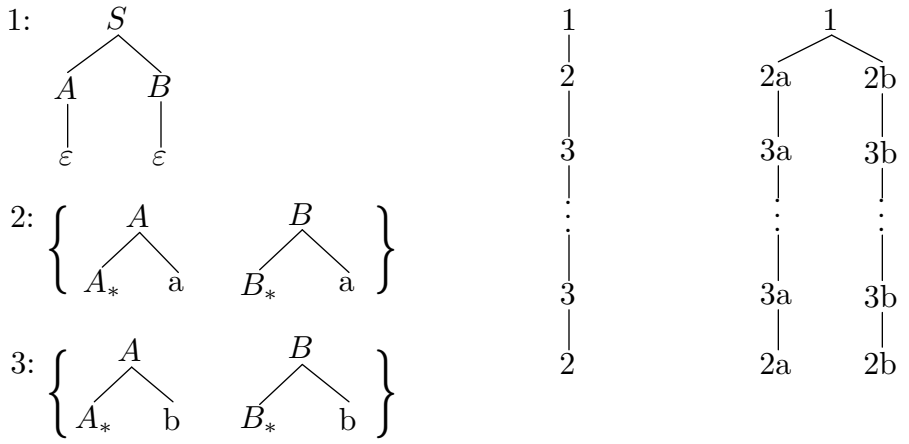


Figure 3.1: An example SL-MCTAG grammar that generates the language ww and associated derivation trees. The derivation tree in the center is the standard derivation tree. The elaborated derivation tree on the right demonstrates an arbitrarily long derivational distance between the trees of a given tree set and their nearest common ancestor. Note that if this grammar is interpreted as a TL-MCTAG grammar only two derivations are possible (for the strings aa and bb).

Tree-Local MCTAG (delayed TL-MCTAG) that uses a derivational notion of locality. In this chapter we introduce two additional derivationally local MCTAG formalisms, restricted non-simultaneous MCTAG (restricted NS MCTAG) and restricted Vector TAG (restricted V-TAG) and demonstrate by construction of parsers how each gives rise to a hierarchy of derivationally local formalisms with a well-defined efficiency penalty for each step of derivational distance permitted.

Although it would be possible to explicitly mark adjunction and substitution sites with link numbers in a derivationally local grammar, the loss of lexical locality means that the link numbers would be global in the grammar and would no longer provide the simplifications in presentation or in complexity analysis that they provide in previous chapters. In this chapter we use the more traditional method of leaving adjunction and substitution sites unmarked and relying on the prohibition on multiple adjunction

and the maximum number of nodes in a given tree of the grammar to constrain the number of operations that occur. This results in one change in our notation for parsers. Rather than associating a potential link ℓ with an item we instead mark the item with one of two diacritics: $\circ$, indicating that an adjunction or substitution is still available at the current node, or $\bullet$, indicating that an adjunction has already taken place at the item's node.

3.2 The Simultaneity Requirement

In addition to lexical locality constraints the definition of MCTAG requires that all trees from a set adjoin simultaneously. In terms of well-formed derivation trees, this amounts to disallowing derivations in which a tree from a given set is the ancestor of a tree from the same tree set. For most linguistic applications of TAG, including all of the analyses presented in Chapters 5 and 6, this requirement seems natural and is strictly obeyed. There are a few applications, including flexible composition and scrambling in free-word order languages that benefit from TAG-based grammars that drop the simultaneity requirement Chiang and Scheffler [2008], Rambow [1994]. From a complexity perspective, however, checking the simultaneity requirement is expensive Kallmeyer [2007]. As a result, it can be seen as advantageous to select a base formalism that does not require simultaneity even if the grammars implemented with it do not make use of that additional freedom.

3.3 Restricted Non-Simultaneous MCTAG

The simplest version of a derivationally local TAG-based formalism is similar to non-local MCTAG. We call this formalism restricted non-simultaneous MCTAG (restricted NS MCTAG). There is no lexical locality requirement at all. In addition, we drop the simultaneity requirement. Thus the only constraint on elementary tree sets is the limit, d, on the derivational distance between the trees in a given set and their nearest common ancestor. Note that if we constrain d to be one, this happens to enforce both the derivational delay limit and the lexical locality requirement of TL-MCTAG.

A CKY-style parser for restricted NS MCTAG with a restriction of d is given in Figure 3.2. The items of this parser contain d lists, Λ^1–Λ^d, called **histories** that record the identities of the trees that have already adjoined in the derivation in order to enforce the locality constraints. The identities of the trees in a tree set that have adjoined in a given derivation are maintained in the histories until all the trees from that set have adjoined. Once the locality constraint is checked for a tree set, the Filter side condition expunges those trees from the histories. A tree is recorded in this history list with superscript i, where i is the derivational distance between the location where the recorded tree adjoined and the location of the current item. The locality constraint is enforced at the point of adjunction or substitution where the history at the limit of the permissible delay must be empty for the operation to succeed.

Goal Item
$$\langle \alpha_0@\varepsilon^\bullet, 0, \_, \_, n, \emptyset, \ldots, \emptyset \rangle \qquad \begin{array}{l}\mathsf{Init}(\alpha_1)\\ \mathsf{Label}(\alpha_0@\varepsilon) = S\\ |\alpha| = 1\end{array}$$

Terminal Axiom
$$\langle \alpha_x@a^\bullet, i, \_, \_, i+1, \emptyset, \ldots, \emptyset \rangle \qquad \mathsf{Label}(\alpha_x@a) = w_{i+1}$$

Empty Axiom
$$\langle \alpha_x@a^\bullet, i, \_, \_, i, \emptyset, \ldots, \emptyset \rangle \qquad \mathsf{Label}(\alpha_x@a) = \varepsilon$$

Foot Axiom
$$\langle \alpha_x@\mathsf{Ft}(\alpha_x)^\circ, p, p, q, q, \emptyset, \ldots, \emptyset \rangle \qquad \mathsf{Aux}(\alpha_x)$$

Unary Complete
$$\frac{\langle \alpha_x@(a\cdot 1)^\bullet, i, j, k, l, \Lambda^1, \ldots, \Lambda^d \rangle}{\langle \alpha_x@a^\circ, i, j, k, l, \Lambda^1, \ldots, \Lambda^d \rangle} \qquad \alpha_x@(a\cdot 2) \text{ undefined}$$

Binary Complete
$$\frac{\begin{array}{c}\langle \alpha_x@(a\cdot 1)^\bullet, i, j, k, l, \Lambda_1^1, \ldots, \Lambda_1^d \rangle\\ \langle \alpha_x@(a\cdot 2)^\bullet, l, j', k', m, \Lambda_2^1, \ldots, \Lambda_2^d \rangle\end{array}}{\langle \alpha_x@a^\circ, i, j\cup j', k\cup k', m, \Lambda^1, \ldots, \Lambda^d \rangle} \qquad \begin{array}{l}\mathsf{Filter}(\Lambda_1^1 \cup \Lambda_2^1, \ldots, \Lambda_1^d \cup \Lambda_2^d)\\ \quad = \Lambda^1, \ldots, \Lambda^d\end{array}$$

Adjoin:
$$\frac{\begin{array}{c}\langle \beta_y@\varepsilon^\bullet, i, p, q, l, \Lambda_1^1, \ldots, \Lambda_1^{d-1}, \emptyset \rangle\\ \langle \alpha_x@a^\circ, p, j, k, q, \Lambda_2^1, \ldots, \Lambda_2^d \rangle\end{array}}{\langle \alpha_x@a^\bullet, i, j, k, l, \Lambda^1, \ldots, \Lambda^d \rangle} \qquad \begin{array}{l}\mathsf{Adj}(\alpha_x@a, \beta_y)\\ \mathsf{Filter}(\Lambda_2^1 \cup \{\beta_y\}, \Lambda_2^2 \cup \Lambda_1^1, \ldots,\\ \quad \Lambda_2^d \cup \Lambda_1^{d-1}) = \Lambda^1, \ldots, \Lambda^d\end{array}$$

Substitute:
$$\frac{\langle \beta_y@\varepsilon^\bullet, i, \_, \_, l, \Lambda_1^1, \ldots, \Lambda_1^{d-1}, \emptyset \rangle}{\langle \alpha_x@a^\bullet, i, \_, \_, l, \Lambda^1, \ldots, \Lambda^d \rangle} \qquad \begin{array}{l}\mathsf{Subst}(\alpha_x@a, \beta_y)\\ \mathsf{Filter}(\{\beta_y\}, \Lambda_1^1, ldots, \Lambda_1^{d-1})\\ \quad = \Lambda^1, \ldots, \Lambda^d\end{array}$$

No Adjoin:
$$\frac{\langle \alpha_x@a^\circ, i, j, k, l, \Lambda^1, \ldots, \Lambda^d \rangle}{\langle \alpha_x@a^\bullet, i, j, k, l, \Lambda^1, \ldots, \Lambda^d \rangle}$$

Figure 3.2: Axioms and inference rules for the CKY algorithm for restricted NS MCTAG with a restriction of d.

3.4 Restricted V-TAG

A Vector-TAG (V-TAG) [Rambow, 1994] is similar to an MCTAG in that the elementary structures are sets (or vectors) of TAG trees. A derivation in a V-TAG is

defined as in TAG. There is no locality requirement or other restriction on adjunction except that if one tree from a vector is used in a derivation, all trees from that vector must be used in the derivation. The trees in a vector may be connected by dominance links between the foot nodes of auxiliary trees and any node in other trees in the vector. All adjunctions must respect the dominance relations in that a node η_1 that dominates a node η_2 must appear on the path from η_2 to the root of the derived tree. The definition of V-TAG is very similar to that of non-local MCTAG as defined by Weir [1988] except that in non-local MCTAG all trees from a tree set are required to adjoin simultaneously.

Restricted V-TAG constrains V-TAG in several ways. First, the dominance chain in each elementary tree vector is required to define a total order over the trees in the vector. This means there is a single **b**ase tree in each vector. Note also that all trees other than the base tree must be auxiliary trees in order to dominate other trees in the vector. The base tree may be either an initial tree or an auxiliary tree. Second, a restricted V-TAG has a restriction level, d, that determines the largest derivational distance that may exist between the base tree and the highest tree in a tree vector in a derivation. Restricted V-TAG differs from restricted NS MCTAG in one important respect: the dominance requirements of restricted V-TAG require that trees from the same set must appear along a single path in the derived tree, whereas in restricted NS MCTAG trees from the same set need not adhere to any dominance relationship in the derived tree.

A CKY-style parser for restricted V-TAG with restriction level d is given in Figure 3.3. Parsing is similar to restricted NS MCTAG in that we have a set of histories

Unary Complete

$$\frac{\langle \alpha_x@(a\cdot 1)^\bullet, i, j, k, l, \Lambda^1, \ldots, \Lambda^d\rangle}{\langle \alpha_x@a^\circ, i, j, k, l, \Lambda^1, \ldots, \Lambda^d\rangle} \qquad \alpha_x@(a\cdot 2) \text{ undefined}$$

Binary Complete

$$\frac{\begin{array}{c}\langle \alpha_x@(a\cdot 1)^\bullet, i, j, k, l, \Lambda_1^1, \ldots, \Lambda_1^d\rangle \\ \langle \alpha_x@(a\cdot 2)^\bullet, l, j', k', m, \Lambda_2^1, \ldots, \Lambda_2^d\rangle\end{array}}{\langle \alpha_x@a^\circ, i, j\cup j', k\cup k', m, \Lambda_1^1\cup\Lambda_2^1, \ldots, \Lambda_1^d\cup\Lambda_2^d\rangle}$$

Adjoin base:

$$\frac{\begin{array}{c}\langle \beta_1@\varepsilon^\bullet, i, p, q, l, \Lambda_1^1, \ldots, \Lambda_1^{d-1}, \emptyset\rangle \\ \langle \alpha_x@a^\circ, p, j, k, q, \Lambda_2^1, \ldots, \Lambda_2^d\rangle\end{array}}{\langle \alpha_x@a^\bullet, i, j, k, l, \Lambda^1, \ldots, \Lambda^d\rangle} \qquad \begin{array}{l}\mathsf{Adj}(\alpha_x@a, \beta_1) \\ \mathsf{Filter}(\Lambda_2^1\cup\{\beta_1\}, \Lambda_2^2\cup\Lambda_1^1, \\ \quad \ldots, \Lambda_2^d\cup\Lambda_1^{d-1}) = \Lambda^1, \ldots, \Lambda^d\end{array}$$

Adjoin non-base:

$$\frac{\begin{array}{c}\langle \beta_y@\varepsilon^\bullet, i, p, q, l, \Lambda_1^1, \ldots, \Lambda_1^{d-1}, \emptyset\rangle \\ \langle \alpha_x@a^\circ, p, j, k, q, \Lambda_2^1, \ldots, \Lambda_2^d\rangle\end{array}}{\langle \alpha_x@a^\bullet, i, j, k, l, \Lambda^1, \ldots, \Lambda^d\rangle} \qquad \begin{array}{l}\mathsf{Adj}(\alpha_x@a, \beta_y) \\ \text{for single } \Lambda_2^i \text{ s.t. } \beta_{y-1}\in\Lambda_2^i, \\ \Lambda_{2'}^i = (\Lambda_2^i\cup\Lambda_1^{i-1}\cup\{\beta_y\}) - \{\beta_{y-1}\} \\ \text{for } \Lambda_2^i \text{ s.t. } \beta_{y-1}\notin\Lambda_2^i, \\ \quad \Lambda_{2'}^i = \Lambda_2^i\cup\Lambda_1^{i-1} \\ \mathsf{Filter}(\Lambda_{2'}^1, \Lambda_{2'}^2\cup\Lambda_1^1, \ldots, \\ \quad \Lambda_{2'}^d\cup\Lambda_1^{d-1}) = \Lambda^1, \ldots, \Lambda^d\end{array}$$

Substitute:

$$\frac{\langle \beta_1@\varepsilon^\bullet, i, \_, \_, l, \Lambda_1^1, \ldots, \Lambda_1^{d-1}, \emptyset\rangle}{\langle \alpha_x@a^\bullet, i, \_, \_, l, \Lambda^1, \ldots, \Lambda^d\rangle} \qquad \begin{array}{l}\mathsf{Subst}(\alpha_x@a, \beta_1) \\ \mathsf{Filter}(\{\beta_1\}, \Lambda_1^1, \ldots, \Lambda_1^{d-1}) \\ \qquad = \Lambda^1, \ldots, \Lambda^d\end{array}$$

No Adjoin:

$$\frac{\langle \alpha_x@a^\circ, i, j, k, l, \Lambda^1, \ldots, \Lambda^d\rangle}{\langle \alpha_x@a^\bullet, i, j, k, l, \Lambda^1, \ldots, \Lambda^d\rangle}$$

Figure 3.3: Inference rules for the CKY algorithm for restricted V-TAG with a restriction of d. Item form, goal item and axioms are omitted because they are identical to those in the restricted NS MCTAG parser.

for each restriction level. However, because of the total order over trees in a vector, the parser only needs to maintain the name of the highest tree from a vector that has been used in the derivation along with its distance from the base tree from that vector. The Filter side condition accordingly expunges trees that are the top tree in

the dominance chain of their tree vector. The side conditions for the Adjoin non-base rule enforce that the dominance constraints are satisfied and that the derivational distance from the base of a tree vector to its currently highest adjoined tree is maintained accurately. We note that in order to allow a non-total ordering of the trees in a vector we would simply have to record all trees in a tree vector in the histories as is done in the restricted NS MCTAG parser.

3.4.1 Limited Delay V-TAG

We also introduce a slightly more constrained version of Restricted V-TAG, called Limited Delay V-TAG, that we make use of as our formalism in the linguistic analyses in Chapter 6. In this version of the formalism all trees must adjoin tree-locally unless they are explicitly marked for delay with a diacritic ($\uparrow$). Only the single tree at the top of the dominance chain may be marked for delay. If a tree set adjoins at a link that has explicit positions for each of the trees in the set, then the delay tree may either adjoin at its link location or it may be delayed to a position higher in the derived tree so long as that position dominates the original link location (which must in turn dominate the other link locations in order for the dominance constraints to be satisfied). If a tree set adjoins at a link that has a link location for each tree other than the tree marked for delay, then the delayed tree must adjoin at a higher location in the derived tree that dominates the root of the tree where it's tree set adjoined. As in Vector TAG, nodes may possess **integrity constraints**. An integrity constraint $\Delta : \tau$ on a node n is violated if the path from a delayed tree to the tree it dominates from a tree set of category τ passes through n.

This alteration effects the parser as follows. If the base tree of tree set β adjoins to tree α, the highest tree in the β dominance chain that is not marked for delay must appear in the first history list of α when the root of α is reached and all operations are complete. This enforces tree locality for all trees other than those marked for delay. The Filter side condition can check this constraint. That only one tree from a given tree set is marked for delay may be checked on the lexicon.

3.5 Delayed TL-MCTAG

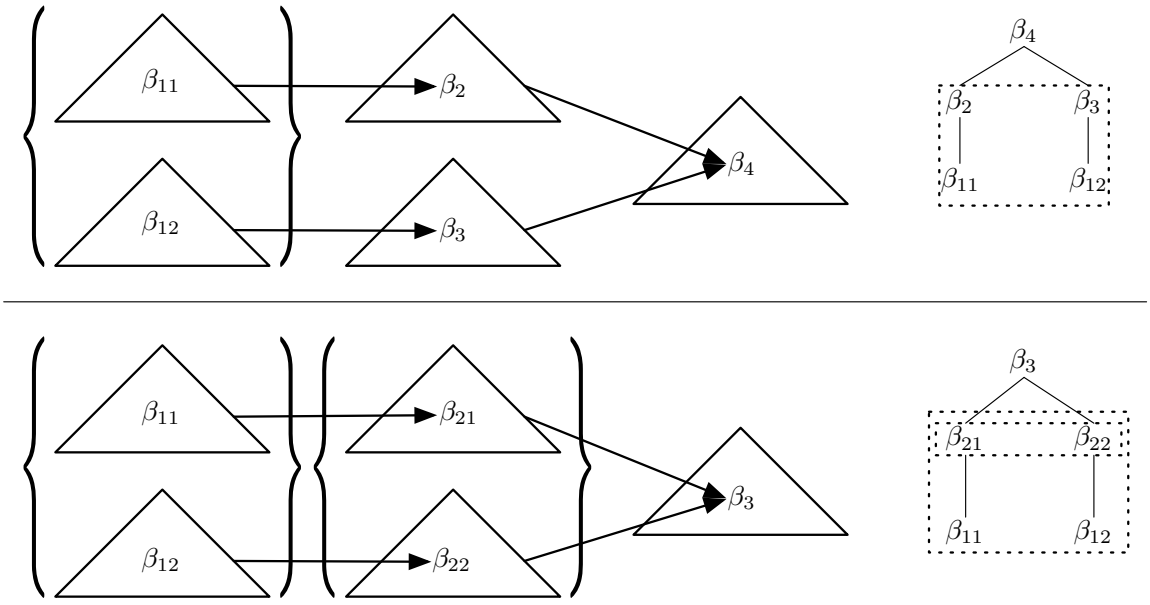


Figure 3.4: Examples of 1-delay (top) and 2-delay (bottom) reproduced from Chiang and Scheffler [2008]. The delays are marked with dashed boxes on the derivation trees.

Chiang and Scheffler [2008] introduce the delayed TL-MCTAG formalism which

makes use of a derivational distance restriction in a somewhat different way. Rather than restricting the absolute distance between the trees of a set and their nearest common ancestor, given a node α in a derivation tree, delayed TL-MCTAG restricts the number of tree sets that are not fully dominated by α. Borrowing directly from Chiang and Scheffler [2008], Figure 3.4 gives two examples.

Parsing delayed TL-MCTAG is not discussed by Chiang and Scheffler [2008] but can be accomplished using a similar CKY-style strategy as in the two parsers above. We present a parser in Figure 3.5. Rather than keeping histories that record derivational distance, we keep an active delay list for each item that records the delays that are active (by recording the identities of the trees that have adjoined) for the tree of which the current node is a part. At the root of each tree the active delay list is filtered using the Filter side condition to remove all tree sets that are fully dominated and the resulting list is checked using the Size to ensure that it contains no more than d distinct tree sets where d is the specified delay for the grammar. The active delays for a given tree are passed to its derivational parent when it adjoins or substitutes.

Delayed TL-MCTAG differs from both of the previous formalisms in that it places no constraint on the length of a delay. On the other hand while the previous formalisms allow unlimited short delays to be pending at the same time, in delayed TL-MCTAG, only a restricted number of delays may be active at once. Similar to restricted V-TAG, there is no simultaneity requirement, so a tree may have another tree from the same set as an ancestor.

Goal Item:
$$\langle \alpha_0@\varepsilon^\bullet, 0, \_, \_, n, \emptyset, \ldots, \emptyset \rangle \qquad \begin{array}{l}\mathsf{Init}(\alpha_1)\\ \mathsf{Label}(\alpha_0@\varepsilon) = S\\ |\alpha| = 1\end{array}$$

Terminal Axiom
$$\langle \alpha_x@a^\bullet, i, \_, \_, i+1, \emptyset, \ldots, \{\alpha_x\} \rangle \qquad \mathsf{Label}(\alpha_x@a) = w_{i+1}$$

Empty Axiom
$$\langle \alpha_x@a^\bullet, i, \_, \_, i, \emptyset, \ldots, \{\alpha_x\} \rangle \qquad \mathsf{Label}(\alpha_x@a) = \varepsilon$$

Foot Axiom
$$\langle \alpha_x@\mathsf{Ft}(\alpha_x)^\circ, p, p, q, q, \emptyset, \ldots, \{\alpha_x\} \rangle \qquad \mathsf{Aux}(\alpha_x)$$

Unary Complete
$$\frac{\langle \alpha_x@(a\cdot 1)^\bullet, i, j, k, l, \Lambda \rangle}{\langle \alpha_x@a^\circ, i, j, k, l, \Lambda \rangle} \qquad \alpha_x@(a\cdot 2) \text{ undefined}$$

Binary Complete
$$\frac{\langle \alpha_x@(a\cdot 1)^\bullet, i, j, k, l, \Lambda_1 \rangle \langle \alpha_x@(a\cdot 2)^\bullet, l, j', k', m, \Lambda_2 \rangle}{\langle \alpha_x@a^\circ, i, j\cup j', k\cup k', m, \Lambda_1 \cup \Lambda_2 \rangle}$$

Adjoin:
$$\frac{\langle \beta_y@\varepsilon^\bullet, i, p, q, l, \Lambda_\beta \rangle \langle \alpha_x@a^\circ, p, j, k, q, \Lambda_\alpha \rangle}{\langle \alpha_x@a^\bullet, i, j, k, l, \Lambda'_\beta \cup \Lambda_\alpha \rangle} \qquad \begin{array}{l}\mathsf{Adj}(\alpha_x@a, \beta_y)\\ \mathsf{Filter}(\Lambda_\beta, \Lambda'_\beta)\\ \mathsf{Size}(\Lambda'_\beta) \le d\end{array}$$

Substitute:
$$\frac{\langle \beta_y@\varepsilon^\bullet, i, \_, \_, l, \Lambda_\beta \rangle}{\langle \alpha_x@a^\bullet, i, \_, \_, l, \Lambda'_\beta \cup \{\alpha_x\} \rangle} \qquad \begin{array}{l}\mathsf{Subst}(\alpha_x@a, \beta_y)\\ \mathsf{Filter}(\Lambda_\beta, \Lambda'_\beta)\\ \mathsf{Size}(\Lambda'_\beta) \le d\end{array}$$

No Adjoin:
$$\frac{\langle \alpha_x@a^\circ, i, j, k, l, \Lambda \rangle}{\langle \alpha_x@a^\bullet, i, j, k, l, \Lambda \rangle}$$

Figure 3.5: Axioms and inference rules for the CKY algorithm for delayed TL-MCTAG with a delay of d.

3.6 Complexity

The complexity of the restricted NS MCTAG and restricted V-TAG parsers presented above depends on the number of possible histories that may appear in an item. For each step of derivational distance permitted between trees of the same set, the

corresponding history permits many more entries. History Λ^1 may contain trees that have adjoined into the same tree as the node of the current item. The number of entries is therefore limited by the number of adjunction sites in that tree, which is in turn limited by the number of nodes in that tree. We will call the maximum number of nodes in a tree in the grammar t. Theoretically, any tree in the grammar could adjoin at any of these adjunction sites, meaning that the number of possible values for each entry in the history is bounded by the size of the grammar $|G|$. Thus the size of Λ^1 is $O(|G|^t)$. For Λ^2 the entries correspond to tree that have adjoined into a tree that has adjoined into the tree of the current item. Thus, for each of the t trees that may have adjoined at a derivational distance of one, there are t more trees that may have adjoined at a derivational distance of two. The size of Λ^2 is therefore $|G|^{t^2}$. The combined size of the histories for a grammar with a delay or restriction of d is therefore $O(|G|^{\sum_{i=1}^{d} t^d})$. Replacing the sum with its closed form solution, we have $O(|G|^{\frac{t^{d+1}-1}{t-1}-1})$ histories.

Using the reasoning about the size of the histories given above, the restricted NS MCTAG parser presented here has a complexity of $O(n^6\,|G|^{1+\frac{t^{d+1}-1}{t-1}})$, where t is as defined above and d is the limit on delay of adjunction. For a tree-local MCTAG, the complexity reduces to $O(n^6\,|G|^{2+t})$. For the linguistic applications that motivate this chapter no delay greater than two is needed, resulting in a complexity of $O(n^6\,|G|^{2+t+t^2})$.

The same complexity analysis applies for restricted V-TAG. However, we can provide a somewhat tighter bound by noting that the **rank**, r of the grammar—how many tree sets adjoin in a single tree–and the **fan out**, f of the grammar—how many

trees may be in a single tree set, are limited by t. That is, a complete derivation containing $|\mathcal{D}|$ tree sets can contain no more than $t\,|\mathcal{D}|$ individual trees and also no more than $rf\,|\mathcal{D}|$ individual trees. In the restricted V-TAG algorithm we maintain only one tree from a tree set in the history at a time, so rather than maintaining $O(t)$ entries in each history, we only need to maintain the smaller $O(r)$ entries.

The complexity of the delayed TL-MCTAG parser depends on the number of possible active delay lists. As above, each delay list may have a maximum of t entries for trees that adjoin directly into it. The restriction on the number of active delays means that the active delay lists passed up from these child nodes at the point of adjunction or substitution can have size no more than d. This results in an additional $td(f-1)$ possible entries in the active delay list, giving a total number active delay lists of $O(|G|^{t(1+d(f-1))})$. Thus the complexity of the parser is $O(n^6\,|G|^{2+t(1+d(f-1))})$.

3.7 Conclusion

Each of the formalisms presented above extends the flexibility of MCTAG beyond that of TL-MCTAG while maintaining, as we have shown herein, complexity much less than that of SL-MCTAG. They permit modeling of flexible composition, analyses of scrambling, the various challenging semantic constructions that we will address in Chapter 6. We conclude that extending locality by constraining derivational distance may be an effective way to add flexibility to MCTAG without losing computational tractability.

Chapter 4

Synchronous Tree-Adjoining Grammar

Recently, the desire to incorporate syntax-awareness into machine translation systems has generated interest in the application of synchronous tree-adjoining grammar (STAG) to this problem [Nesson et al., 2006, Chiang and Rambow, 2006]. In a parallel development, interest in incorporating semantic computation into the TAG framework has led to the use of STAG for this purpose [Nesson and Shieber, 2007, Han, 2006a,b, Nesson and Shieber, 2006]. Although STAG does not increase the expressivity of the underlying formalisms [Shieber, 1994], STAG parsing is known to be NP-hard due to the potential for intertwined correspondences between the linked nonterminal symbols in the elementary structures [Satta, 1992, Weir, 1988]. Without efficient algorithms for processing it, its potential for use in machine translation and TAG semantics systems is limited.

Given a particular grammar, the polynomial degree of efficient STAG parsing

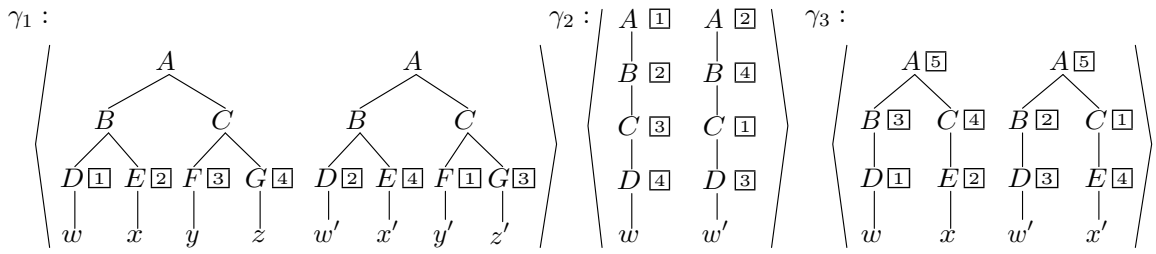


Figure 4.1: Example of intertwined links that cannot be binarized. No two links can be isolated in both trees in a tree pair. Note that in tree pair γ_1, any set of three links may be isolated while in tree pair γ_2, no group of fewer than four links may be isolated. In γ_3 no group of links smaller than four may be isolated.

algorithms depends directly on a factor k: the maximum number of intertwined correspondences that appear within a single elementary structure of the grammar. This is illustrated by the tree pairs given in Figure 4.1 in which no two numbered links may be isolated. (By "isolated", I mean that the links can be contained in a fragment of the tree that contains no other links and dominates only one branch not contained in the fragment. A precise definition is given in Section 4.2.)

An analogous problem has long been known to exist for synchronous context-free grammars (SCFG) [Aho and Ullman, 1969]. The task of producing efficient parsers for SCFG has recently been addressed by binarization or k-arization of SCFG grammars that produce equivalent grammars in which the degree of intertwined correspondences between synchronous rules has been minimized [Zhang and Gildea, 2007, Zhang et al., 2006, Gildea et al., 2006]. The methods for k-arization of SCFG cannot be directly applied to STAG because of the additional complexity introduced by the expressivity-increasing adjunction operation of TAG. In SCFG, where substitution is the only available operation and the depth of elementary structures is limited to one, the k-arization problem reduces to analysis of permutations of strings of nonterminal

symbols. In STAG, however, the arbitrary depth of the elementary structures and the lack of restriction to contiguous strings of nonterminals introduced by adjunction substantially complicate the task.

In this chapter I offer the first algorithm addressing this problem for the STAG case. I present a compile-time algorithm for transforming a STAG into a strongly-equivalent STAG that optimally minimizes k across the grammar. This is a critical minimization because k is related to the feature of the grammar that appears in the exponent of the complexity of parsing algorithms for STAG. The complexity for a standard STAG parser implementation is $\mathcal{O}(n^{4\cdot(r+1)} \cdot |G|^{(r+1)})$ where r is the rank of the grammar (see Section 4.1). When k is minimized and the grammar is factorized, r is equal to k. Thus, by minimizing k, the worst-case complexity of a parser instantiated for a particular grammar is optimized. The k-arization algorithm performs in $\mathcal{O}(|G| + |Y| \cdot L_G^3)$ time where L_G is the maximum number of links in any single synchronous tree pair in the grammar and Y is the set of synchronous tree pairs of G. By comparison, a baseline algorithm performing exhaustive search requires $\mathcal{O}(|G| + |Y| \cdot L_G^6)$ time.[1]

The remainder of this chapter proceeds as follows. In Section 4.1 I provide a brief introduction to the STAG formalism. I present the k-arization algorithm in Section 4.2 and an analysis of its complexity in Section 4.3. I prove the correctness of the algorithm in Section 4.4.

[1]In a synchronous tree with L links, there are $\mathcal{O}(L^4)$ pairs of valid fragments, and it takes $\mathcal{O}(L)$ time to check whether the two components in a pair have the same set of links. Once the synchronous fragment with the smallest number of links is excised, the above process must be iterated at most L times, resulting in overall time $\mathcal{O}(L_G^6)$.

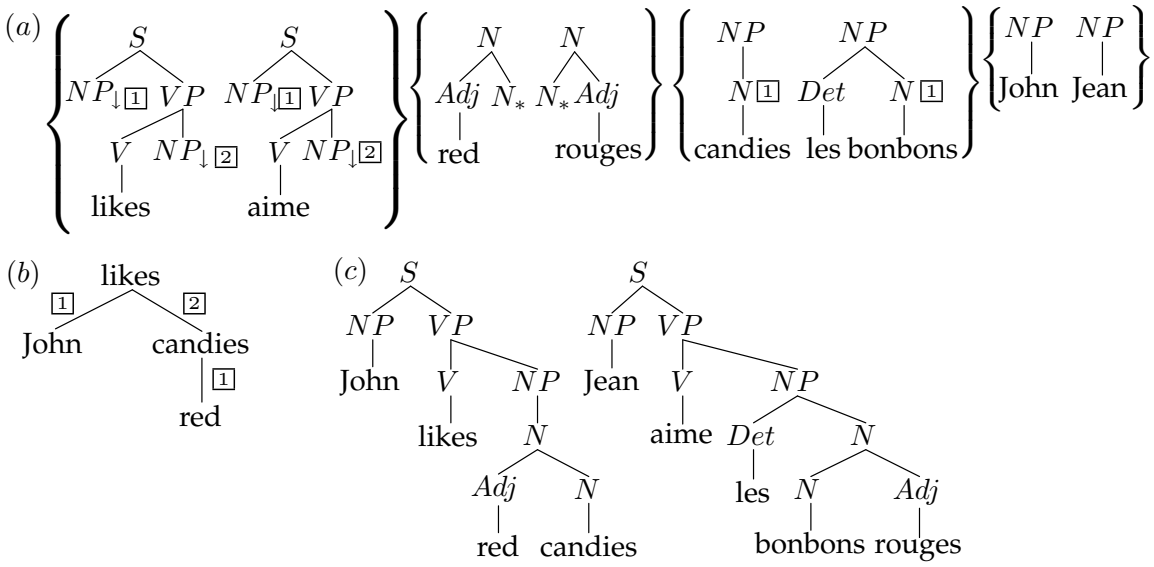


Figure 4.2: An example STAG derivation of the English/French sentence pair "John likes red candies"/"Jean aime les bonbons rouges". The figure is divided as follows: (a) the STAG grammar, (b) the derivation tree for the sentence pair, and (c) the derived tree pair for the sentences.

4.1 Synchronous Tree-Adjoining Grammar

As introduced in Chapter 1, in a synchronous TAG (STAG) the elementary structures are ordered pairs of TAG trees, with a linking relation specified over pairs of nonterminal nodes. Each link has two locations, one in the left tree in a pair and the other in the right tree. An example of an STAG derivation including both substitution and adjunction is given in Figure 4.2.

Although it is not immediately obvious, STAG is a special case of SL-MCTAG. To see this, consider how one might convert an arbitrary STAG to an SL-MCTAG. This can be accomplished simply by adding a single new tree set of length 1 containing a tree that is rooted in a new start symbol and with a fringe consisting of two substitution nodes labeled with the start symbols of the source and target grammars of

the STAG and annotated with a matching link number. For any valid STAG derivation a final step is added in which the source and target start trees are substituted into the new start tree to produce a complete and equivalent SL-MCTAG derivation. The change in formalism does not make any new derivations possible as long as the non-terminal symbols of the source and target grammars are renamed apart if they are not already disjoint.

It is worth noting, however, that this SL-MCTAG derivation has some very particular characteristics. First, it might be called a *strictly* set-local derivation in that all tree sets except the new start tree set have length 2 and, if each tree in the tree set is indexed according to whether it is a source or target tree, all operations will occur only between trees that have the same index. That is, source trees will only combine with source trees and target trees will only combine with target trees. Strictness does not generally hold for SL-MCTAG derivations. Second, an SL-MCTAG grammar may contain tree sets with length greater than 2 and correspondingly it will also contain links with more than two link locations.

Although SL-MCTAG parsing is known to be PSPACE-complete, these differences between STAG and SL-MCTAG result in some efficiency gains in parsing STAG. The strictness of STAG essentially enforces that STAG operates under the vector definition rather than the set definition. That is, because each link has exactly one location in each of the synchronized grammars and that link location can only be used by a tree from the same grammar, there is no possibility to permute how links are used. In addition, in STAG the fan out of the grammar, f, is limited to 2. As a result, STAG parsing can be accomplished using the standard method in $O(n^{4(r+1)}\,|G|^{(r+1)})$.

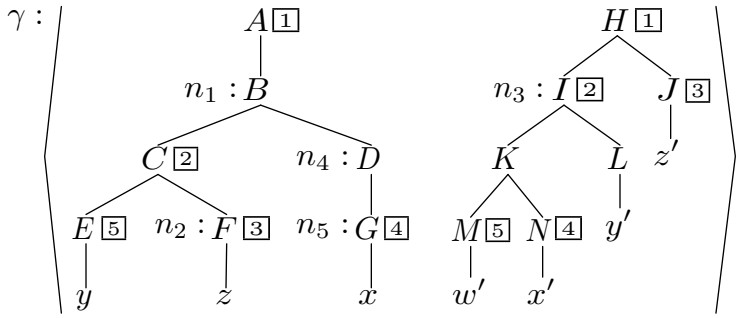


Figure 4.3: A synchronous tree pair containing fragments $\alpha_L = \gamma_L(n_1, n_2)$ and $\alpha_R = \gamma_R(n_3)$. Since $\mathsf{links}(n_1, n_2) = \mathsf{links}(n_3) = \{\boxed{2}, \boxed{4}, \boxed{5}\}$, synchronous fragment $\alpha = \langle \alpha_L, \alpha_R \rangle$ is valid. Note also that node n_3 is a maximal node and node n_5 is not. $\sigma(n_1) = \boxed{2}\boxed{5}\boxed{5}\boxed{3}\boxed{3}\boxed{2}\boxed{4}\boxed{4}$; $\sigma(n_3) = \boxed{2}\boxed{5}\boxed{5}\boxed{4}\boxed{4}\boxed{2}$.

4.2 k-arization Algorithm

In this section I present the k-arization algorithm. This algorithm factorizes an STAG grammar into a strongly equivalent STAG grammar in which each tree pair contains a set of links that cannot be factorized further. I begin by presenting several definitions that are of use in the algorithm.

For a synchronous tree pair $\gamma = \langle \gamma_L, \gamma_R \rangle$, a **fragment** of γ_L (or γ_R) is a complete subtree rooted at some node n of γ_L, written $\gamma_L(n)$, or else a subtree rooted at n with a gap at node n', written $\gamma_L(n, n')$; see Figure 4.3 for an example. $\mathsf{links}(n)$ and $\mathsf{links}(n, n')$ are written to denote the set of links of $\gamma_L(n)$ and $\gamma_L(n, n')$, respectively. When the root or gap nodes of some fragment α_L are unknown, $\mathsf{links}(\alpha_L)$ is written.

A set of links Λ from γ can be **isolated** if there exist fragments α_L and α_R of γ_L and γ_R, respectively, both with links Λ. If this is the case, a synchronous fragment $\alpha = \langle \alpha_L, \alpha_R \rangle$ can be constructed. The goal of the algorithm is to decompose γ into synchronous fragments such that the maximum number of links of a synchronous frag-

1	1	0	0	0	0	0	0	0	0	1
2	0	1	0	0	0	0	1	0	1	0
5	0	0	1	1	0	0	0	0	0	0
5	0	0	1	1	0	0	0	0	0	0
3	0	0	0	0	0	0	0	1	1	0
3	0	0	0	0	0	0	0	1	1	0
2	0	1	0	0	0	0	1	0	0	0
4	0	0	0	0	1	1	0	0	0	0
4	0	0	0	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	1
	1	2	5	5	4	4	2	3	3	1

Figure 4.4: Table π with synchronous fragment $\langle \gamma_L(n_1, n_2),\ \gamma_R(n_3) \rangle$ from Figure 4.3 highlighted.

ment is kept to a minimum, and γ can be obtained from the synchronous fragments by means of the usual substitution and adjunction operations. In order to simplify the presentation of the algorithm it is assumed without any loss of generality that all elementary trees of the source STAG have nodes with at most two children.

4.2.1 Maximal nodes

A node n of γ_L (or γ_R) is called **maximal** if (i) $\mathsf{links}(n) \neq \emptyset$, and (ii) it is either the root node of γ_L or, for its parent node n', $\mathsf{links}(n') \neq \mathsf{links}(n)$. Note that for every node n' of γ_L such that $\mathsf{links}(n') \neq \emptyset$ there is always a unique maximal node n such that $\mathsf{links}(n') = \mathsf{links}(n)$. Thus, for the purpose of the algorithm, one need only look at maximal nodes as places for excising tree fragments. It can be shown that the number of maximal nodes M_n in a subtree $\gamma_L(n)$ always satisfies $|\mathsf{links}(n)| \leq M_n \leq 2 \times |\mathsf{links}(n)| - 1$.

Let n be some node of γ_L, and let $l(n)$ be the (unique) link impinging on n if

such a link exists, and $l(n) = \varepsilon$ otherwise. Let n be associated with a string $\sigma(n)$, defined by a pre- and post-order traversal of fragment $\gamma_L(n)$. The symbols of $\sigma(n)$ are the links in $\mathsf{links}(n)$, viewed as atomic symbols. Given a node n with p children $n_1, \ldots, n_p$, $0 \leq p \leq 2$, define $\sigma(n) = l(n)\,\sigma(n_1) \cdots \sigma(n_p)\,l(n)$. See again Figure 4.3 for an example. Note that $|\sigma(n)| = 2 \times |\mathsf{links}(n)|$.

4.2.2 Excision of Synchronous Fragments

Although it would be possible to excise synchronous fragments without creating new nonterminal nodes, for clarity a simple tree transformation that leaves existing nodes intact is used when a fragment is excised. A schematic depiction is given in Figure 4.5. In the figure the excision process is demonstrated on one half of a synchronous fragment: $\gamma_L(n_1, n_2)$ is excised to form two new trees. The excised tree is not processed further. In the excision process the root and gap nodes of the original tree are not altered. The material between them is replaced with a single new node with a fresh nonterminal symbol and a fresh link number. This nonterminal node and link form the adjunction or substitution site for the excised tree. Note that any link impinging on the root node of the excised fragment is by convention included in the fragment and any link impinging on the gap node is not.

To regenerate the original tree, the excised fragment can be adjoined or substituted back into the tree from which it was excised. The new nodes that were generated in the excision may be removed and the original root and gap nodes may be merged back together retaining any impinging links, respectively. Note that if there was a link on either the root or gap node in the original tree, it is not lost or duplicated in

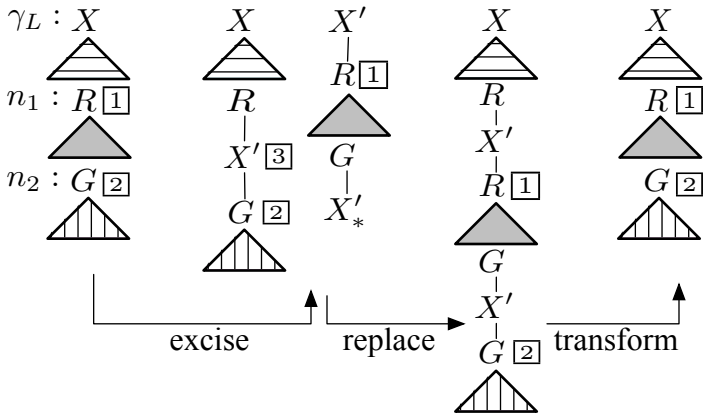


Figure 4.5: A diagram of the tree transformation performed when fragment $\gamma_L(n_1, n_2)$ is removed. Shaded or patterned triangles represent segments of the tree that may comprise multiple nodes and may contain links.

the process.

4.2.3 Method

Let n_L and n_R be the root nodes of trees γ_L and γ_R, respectively. It is clear that $\mathsf{links}(n_L) = \mathsf{links}(n_R)$, and $|\sigma(n_L)| = |\sigma(n_R)|$, the second string being a rearrangement of the occurrences of symbols in the first one. The main data structure of the algorithm is a Boolean matrix π of size $|\sigma(n_L)| \times |\sigma(n_L)|$, whose rows are addressed by the occurrences of symbols in $\sigma(n_L)$, in the given order, and whose columns are similarly addressed by $\sigma(n_R)$. For occurrences of links $\boxed{j_1}, \boxed{j_2}$, the element of π at a row addressed by j_1 and a column addressed by j_2 is 1 if $j_1 = j_2$, and 0 otherwise. Thus, each row and column of π has exactly two non-zero entries. See Figure 4.4 for an example.

For a maximal node n_1 of γ_L, let $\pi(n_1)$ denote the stripe of adjacent rows of π

```
1: Function KARIZE(G) {G a binary STAG}
2: G' ← STAG with empty set of synch trees;
3: for all γ = ⟨γ_L, γ_R⟩ in G do
4:    init π and B;
5:    while B ≠ ∅ do
6:       α_L ← next fragment from B;
7:       α_R ← ISOLATE(α_L, π, γ_R);
8:       if α_R ≠ null then
9:          add ⟨α_L, α_R⟩ to G';
10:         excise ⟨α_L, α_R⟩ from γ;
11:         update π and B;
12:   add γ to G';
13: return G'
```

Figure 4.6: Main algorithm.

addressed by substring $\sigma(n_1)$ of $\sigma(n_L)$. If n_1 dominates n_2 in γ_L, let $\pi(n_1, n_2)$ denote the rows of π addressed by $\sigma(n_1)$ but not by $\sigma(n_2)$. This forms a pair of horizontal stripes in π. For nodes n_3, n_4 of γ_R, similarly $\pi(n_3)$ and $\pi(n_3, n_4)$ are defined as vertical stripes of adjacent columns. See again Figure 4.4.

The algorithm is reported in Figure 4.6. For each synchronous tree pair $\gamma = \langle \gamma_L, \gamma_R \rangle$ from the input grammar, an agenda B is maintained with all candidate fragments α_L from γ_L having at least two links. These fragments are processed greedily in order of increasing number of links. The function ISOLATE(), described in more detail below, looks for a right fragment α_R with the same links as α_L. Upon success, the synchronous fragment $\alpha = \langle \alpha_L, \alpha_R \rangle$ is added to the output grammar. Furthermore, α is excised from γ and data structures π and B are updated. The above process is iterated until B becomes empty. Section 4.4 shows that this greedy strategy is sound and complete.

The function ISOLATE() is specified in Figure 4.7. It takes as input a left fragment

1: **Function** ISOLATE(α_L, π, γ_R)
2: select $n \in \gamma_R$ such that $\sigma(n)$ is the narrowest string within $\sigma(n_R)$ including left/right boundaries of α_L in π;
3: **if** $|\sigma(n)| = 2 \times |\mathsf{links}(\alpha_L)|$ **then**
4: **return** $\gamma_R(n)$;
5: select $n' \in \gamma_R$ such that $\sigma(n')$ is the gap string within $\sigma(n)$ for which $\mathsf{links}(n) - \mathsf{links}(n') = \mathsf{links}(\alpha_L)$;
6: **if** n' is not defined **then**
7: **return null**; {more than one gap}
8: **return** $\gamma_R(n, n')$;

Figure 4.7: Find synchronous fragment.

α_L, which is associated with one or two horizontal stripes in π, depending on whether α_L has a gap node or not. The left boundary of α_L in π is the index j_1 of the column containing the leftmost occurrence of a 1 in the horizontal stripes associated with α_L. Similarly, the right boundary of α_L in π is the index j_2 of the column containing the rightmost occurrence of a 1 in these stripes. It retrieves the narrowest substring $\sigma(n)$ of $\sigma(n_R)$ that spans over indices j_1 and j_2. This means that n is the lowest node from γ_R such that the links of α_L are a subset of the links of $\gamma_R(n)$.

If the condition at line 3 is satisfied, all of the matrix entries of value 1 that are found from column j_1 to column j_2 fall within the horizontal stripes associated with α_L. In this case it reports the right fragment $\alpha_R = \gamma_R(n)$. Otherwise, it checks whether the 1 entries that fall outside of the two horizontal stripes in between columns j_1 and j_2 occur within adjacent columns, say from column $j_3 \geq j_1$ to column $j_4 \leq j_2$. In this case, it then checks whether there exists some node n' such that the substring of $\sigma(n)$ from position j_3 to j_4 is an occurrence of string $\sigma(n')$. This means that n' is the gap node, and it reports the right fragment $\alpha_L = \gamma_R(n, n')$. See again Figure 4.4.

4.3 Complexity

This section presents an implementation of the algorithm of section 4.2 resulting in time complexity $\mathcal{O}(|G| + |Y| \cdot L_G^3)$, where Y is the set of synchronous tree pairs of G and L_G is the maximum number of links in a synchronous tree pair in Y.

Consider a synchronous tree pair $\gamma = \langle \gamma_L, \gamma_R \rangle$ with L links. If M is the number of maximal nodes in γ_L or γ_R, then $M = \mathcal{O}(L)$ (Section 4.2.1). The sparse table π is implemented in $\mathcal{O}(L)$ space, recording for each row and column the indices of its two non-zero entries. It is assumed that it is possible to go back and forth between maximal nodes n and strings $\sigma(n)$ in constant time. Here each $\sigma(n)$ is represented by its boundary positions within $\sigma(n_L)$ or $\sigma(n_R)$, n_L and n_R the root nodes of γ_L and γ_R, respectively.

At line 2 of the function ISOLATE() (Figure 4.7) the left and right boundaries are retrieved by scanning the rows of π associated with input fragment α_L. Node n is then retrieved by visiting all maximal nodes of γ_L spanning these boundaries. Under the above assumptions, this can be done in time $\mathcal{O}(L)$. Line 5 is implemented in a similar way, resulting in overall run time $\mathcal{O}(L)$ for function ISOLATE().

The function KARIZE() (Figure 4.6) uses buckets B_i, $1 \leq i \leq L$, where each B_i stores the candidate fragments α_L with $|\mathsf{links}(\alpha_L)| = i$. To populate these buckets, it first processes fragments $\gamma_L(n)$ by visiting bottom up the maximal nodes of γ_L. The quantity $|\mathsf{links}(n)|$ is computed from the quantities $|\mathsf{links}(n_i)|$, where n_i are the highest maximal nodes dominated by n. (There are at most two such nodes.) Fragments $\gamma_L(n, n')$ can then be processed using the relation $|\mathsf{links}(n, n')| = |\mathsf{links}(n)| - |\mathsf{links}(n')|$. In this way each fragment is processed in constant time, and population of all the

buckets takes $\mathcal{O}(L^2)$ times.

Now consider the while loop at lines 5 to 11 in function KARIZE(). For a synchronous tree pair γ, the loop iterates once for each candidate fragment α_L in some bucket. There are a total of $\mathcal{O}(L^2)$ iterations, since the initial number of candidates in the buckets is $\mathcal{O}(L^2)$, and the possible updating of the buckets after a synchronous fragment is removed does not increase the total size of all the buckets. If the links in α_L cannot be isolated, one iteration takes time $\mathcal{O}(L)$ (the call to function ISOLATE()). If the links in α_L can be isolated, then π must be restructured and the buckets repopulated. The former can be done in time $\mathcal{O}(L)$ and the latter takes time $\mathcal{O}(L^2)$, as already discussed. Crucially, the updating of π and the buckets takes place no more than $L - 1$ times. This is because each time a synchronous fragment is excised, the number of links in γ is reduced by at least one.

In conclusion, function KARIZE() takes time $\mathcal{O}(L^3)$ for each synchronous tree γ, and the total running time is $\mathcal{O}(|G| + |Y| \cdot L_G^3)$, where Y is the set of synchronous tree pairs of G. The term $|G|$ accounts for the reading of the input, and dominates the complexity of the algorithm only in case there are very few links in each synchronous tree pair.

4.4 Proof of Correctness

The algorithm presented in the previous sections produces an optimal k-arization for the input grammar. In this section I sketch a proof of correctness of the strategy

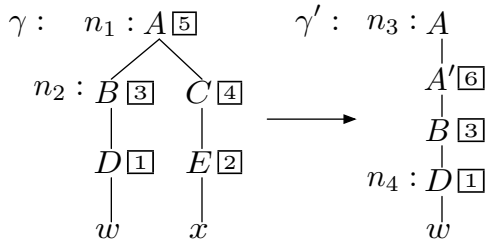


Figure 4.8: In γ links 3 and 5 cannot be isolated because the fragment would have to contain two gaps. However, after the removal of fragment $\gamma(n_1, n_2)$, an analogous fragment $\gamma'(n_3, n_4)$ may be removed.

employed by the algorithm.[2]

The k-arization strategy presented above is greedy in that it always chooses the excisable fragment with the smallest number of links at each step and does not perform any backtracking. It must therefore be shown that this process cannot result in a non-optimal solution. If fragments could not overlap each other, this would be trivial to show because the excision process would be confluent. If all overlapping fragments were cases of complete containment of one fragment within another, the proof would also be trivial because the smallest-to-largest excision order would guarantee optimality. However, it is possible for fragments to partially overlap each other, meaning that the intersection of the set of links contained in the two fragments is non-empty and the difference between the set of links in one fragment and the other is also non-empty. Overlapping fragment configurations are given in Figure 4.9 and discussed in detail below.

The existence of partially overlapping fragments complicates the proof of optimal-

[2]Note that the soundness of the algorithm can be easily verified from the fact that the removal of fragments can be reversed by performing standard STAG adjunction and substitution operations until a single STAG tree pair is produced. This tree pair is trivially homomorphic to the original tree pair and can easily be mapped to the original tree pair.

ity for two reasons. First, the excision of a fragment α that is partially overlapped with another fragment β necessarily precludes the excision of β at a later stage in the excision process. Second, the removal of a fragment may cause a previously non-isolatable set of links to become isolatable, effectively creating a new fragment that may be advantageous to remove. This is demonstrated in Figure 4.8. These possibilities raise the question of whether the choice between removing fragments α and β may have consequences at a later stage in the excision process. The proof below demonstrates that this choice cannot affect the k found for a given grammar. I begin by sketching the proof of a lemma that shows that removal of a fragment β that partially overlaps another fragment α always leaves an analogous fragment that may be removed.

4.4.1 Definition: validity preserving

Consider a STAG tree pair γ containing the set of links Λ and two synchronous fragments α and β with α containing links $\mathsf{links}(\alpha)$ and β containing $\mathsf{links}(\beta)$ and $\mathsf{links}(\alpha) \subsetneq \Lambda, \mathsf{links}(\beta) \subsetneq \Lambda$.

If α and β do not overlap, the removal of β is trivially defined as validity preserving with respect to α. If α and β overlap, removal of β from γ is **validity preserving** with respect to α if after the removal there exists a valid synchronous fragment (containing at most one gap on each side) that contains all and only the links $(\mathsf{links}(\alpha) - \mathsf{links}(\beta)) \cup \{\boxed{x}\}$ where $\boxed{x}$ is the new link added to γ.

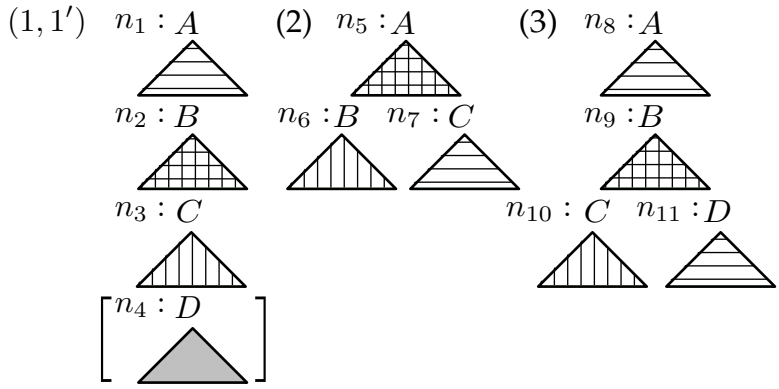


Figure 4.9: The four possible configurations of overlapped fragments within a single tree. For type 1, let $\alpha = \gamma(n_1, n_3)$ and $\beta = \gamma(n_2, n_4)$. The roots and gaps of the fragments are interleaved. For type 1′, let $\alpha = \gamma(n_1, n_3)$ and $\beta = \gamma(n_2)$. The root of β dominates the gap of α. For type 2, let $\alpha = \gamma(n_5, n_6)$ and $\beta = \gamma(n_5, n_7)$. The fragments share a root and have gap nodes that do not dominate each other. For type 3 let $\alpha = \gamma(n_8, n_{10})$ and $\beta = \gamma(n_9, n_{11})$. The root of α dominates the root of β, both roots dominate both gaps, but neither gap dominates the other.

4.4.2 Lemma: Fragment Excision is Validity Preserving

The lemma presented here proves that removal of any synchronous fragment from an STAG tree pair is validity preserving with respect to all of the other synchronous fragments in the tree pair. It suffices to show that for two arbitrary synchronous fragments α and β, the removal of β is validity preserving with respect to α. This is shown by examination of the possible configurations of α and β.

Consider the case in which β is fully contained within α. In this case $\mathsf{links}(\beta) \subsetneq \mathsf{links}(\alpha)$. The removal of β leaves the root and gap of α intact in both trees in the pair, so it remains a valid fragment. All of the links from $\mathsf{links}(\beta)$ are in $\mathsf{links}(\alpha)$, so $\mathsf{links}(\alpha) \cap \mathsf{links}(\beta) = \mathsf{links}(\beta)$. The new link is added at the new node inserted where β was removed. Since β is fully contained within α, this node is below the

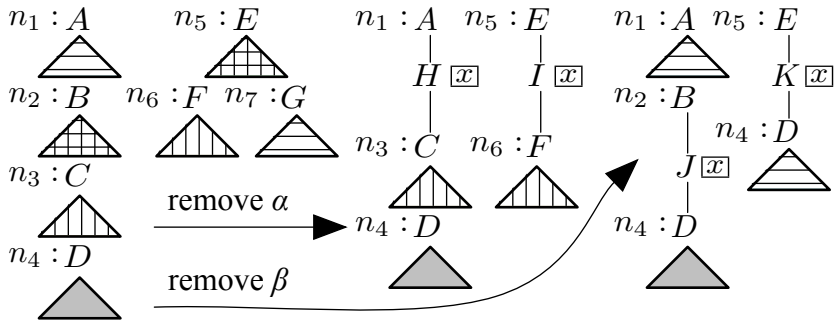


Figure 4.10: Removal from a tree pair γ containing type 1–type 2 fragment overlap. The fragment α is represented by the horizonal-lined pieces of the tree pair. The fragment β is represented by the vertical-lined pieces of the tree pair. Cross-hatching indicates the overlapping portion of the two fragments.

root of α but not below its gap. Thus, the removal process leaves α with the links $(\mathsf{links}(\alpha) - \mathsf{links}(\beta)) \cup \{\boxed{x}\}$, where $\boxed{x}$ is the link added in the removal process; the removal is validity preserving.

Synchronous fragments may partially overlap in several different ways. There are four possible configurations for an overlapped fragment within a single tree, depicted in Figure 4.9. These different single-tree overlap types can be combined in any way to form valid synchronous fragments. I consider two illustrative cases here. Note that in the diagrams that follow, patterned or shaded triangles represent segments of the tree that contain multiple nodes and at least one link. Where the pattern or shading corresponds across trees in a tree pair, the set of links contained within those triangles are equivalent.

An example of removing fragments from a tree set containing type 1–type 2 overlapped fragments is given in Figure 4.10. Let $\alpha = \langle \gamma_L(n_1, n_3), \gamma_R(n_5, n_6) \rangle$. Let $\beta = \langle \gamma_L(n_2, n_4), \gamma_R(n_5, n_7) \rangle$. If α is removed, the validity preserving fragment for

β is $\langle\gamma'_L(n_1, n_4), \gamma'_R(n_5)\rangle$. It contains the links in the vertical-lined part of the tree and the new link $\boxed{x}$. This forms a valid fragment because both sides contain at most one gap and both contain the same set of links. In addition, it is validity preserving for β because it contains exactly the set of links that were in $\mathsf{links}(\beta)$ and not in $\mathsf{links}(\alpha)$ plus the new link $\boxed{x}$. If we instead choose to remove β, the validity preserving fragment for α is $\langle\gamma'_L(n_1, n_4), \gamma'_R(n_5)\rangle$. The links in each side of this fragment are the same, each side contains at most one gap, and the set of links is exactly the set left over from $\mathsf{links}(\alpha)$ once $\mathsf{links}(\beta)$ is removed plus the newly generated link $\boxed{x}$.

An example of removing fragments from a tree set containing type 1′–type 3 (reversed) overlapped fragments is given in Figure 4.11. If α is removed, the validity preserving fragment for β is $\langle\gamma'_L(n_1), \gamma'_R(n_4)\rangle$. If β is removed, the validity preserving fragment for α is $\langle\gamma'_L(n_1, n_8), \gamma'_R(n_4)\rangle$.

Similar reasoning follows straightforwardly for all remaining types of overlapped fragments.

4.4.3 Proof Sketch: Smallest-First Removal of Fragments is Optimal

Consider a decision point at which a choice is made about which fragment to remove. Call the size of the smallest fragments at this point m, and let the set of fragments of size m be X with $\alpha, \beta \in X$.

There are two cases to consider. First, consider two partially overlapped fragments $\alpha \in X$ and $\delta \notin X$. Note that $|\mathsf{links}(\alpha)| < |\mathsf{links}(\delta)|$. Validity preservation of α with respect to δ guarantees that δ or its validity preserving analog will still be available

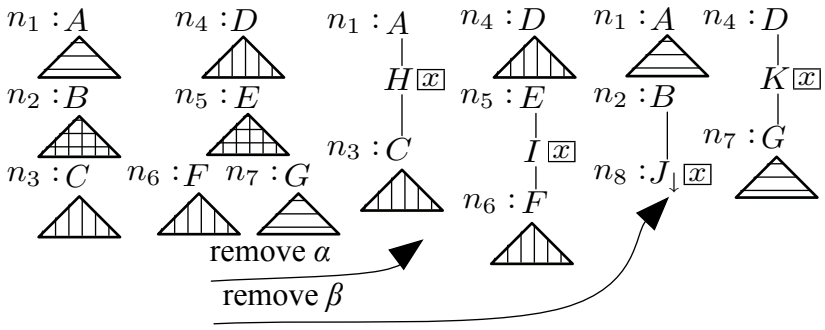


Figure 4.11: Removal from a tree pair γ containing a type 1′–type 3 (reversed) fragment overlap. The fragment α is represented by the horizontal lined pieces of the tree pair. The fragment β is represented by the vertical-lined pieces of the tree pair. Cross-hatching indicates the overlapping portion of the two fragments.

for excision after α is removed. Excising δ increases k more than excising α or any fragment that removal of α will lead to before δ is considered. Thus, removal of δ cannot result in a smaller value for k if it is removed before α rather than after α.

Second, consider two partially overlapped fragments $\alpha, \beta \in X$. First note that due to the validity preservation lemma, an arbitrary choice between the fragments in X does not affect the opportunity to remove other fragments (or their validity preserving analogs) in that set at a later point. Removal of fragment α cannot increase the size of any remaining fragment.

Removal of α or β may generate new fragments that were not previously valid and may reduce the size of existing fragments that it overlaps. In addition, removal of α may lead to availability of smaller fragments at the next removal step than removal of β (and vice versa). However, since removal of either α or β produces a k of size at least m, the later removal of fragments of size less than m cannot affect the k found by the algorithm. Finally, note that due to validity preservation, removal of any of these smaller fragments will still permit removal of all currently existing fragments

or their analogs at a later step in the removal process.

If the removal of α generates a new fragment δ of size larger than m all remaining fragments in X (and all others smaller than δ) will be removed before δ is considered. Therefore, if removal of β generates a new fragment smaller than δ, the smallest-first strategy will properly guarantee its removal before δ.

4.5 Conclusion

The potential for the use of STAG in machine translation and other natural-language processing tasks has been limited by the challenges of processing it efficiently. The difficulty in parsing STAG stems directly from the factor k which indicates the degree to which the correspondences are intertwined within the elementary structures of the grammar. The algorithm presented in this chapter is the first method available for k-arizing a synchronous TAG grammar into an equivalent grammar with an optimal value for k. The algorithm operates offline and requires only $\mathcal{O}(|G| + |Y| \cdot L_G^3)$ time. Both the derivation trees and derived trees produced are trivially homomorphic to those that are produced by the original grammar.

Part II

Linguistic Applications

Chapter 5

Modeling the Syntax-Semantics Interface using Synchronous TAG

Syntactic and semantic structure are so closely related as to suggest a direct correspondence between them. However, efforts to draw such a connection quickly encounter the many situations in which they are divergent, such as in the case of quantifiers and other scope taking elements that appear in vastly different locations in the syntactic structure than they do in the semantic structure. In this chapter we propose modeling the syntactic-semantic interface by formally connecting syntactic and semantic grammars using grammar synchronization. Grammar synchronization captures the tight coupling between syntax and semantics by enforcing the constraint that both the syntax and semantics of a sentence must share the same derivational structure. Intuitively, the derivational structure forms a highly appropriate interface between the syntax and semantics of natural language. It encapsulates the minimal information necessary to know which lexical items have combined with each other,

defining the basic semantic relationships present in a sentence. However, derived syntactic and semantic structures are often starkly divergent. Synchronization through the derivational structure provides sufficient flexibility to support this divergence because it permits great difference in the syntactic and semantic lexicons.

This chapter makes the first attempt to thoroughly explore the proposal described in Shieber and Schabes [1990] for using synchronous TAG to model the syntactic-semantic interface by taking the tree pairs to represent a syntactic analysis synchronized with a semantic analysis. TAG is particularly well-suited as a base formalism to synchronize for several reasons. First, it naturally models the basic linguistic operations of argument substitution and optional modification using its operations substitution and adjunction. Second, it is widely used for analysis of syntactic phenomena by computational linguistics researchers, research into how to incorporate semantic interpretation into the TAG framework has lagged behind. We offer an alternative to other methods also currently under development. Third, it is potentially a computationally appealing formalism if a variant that can be processed efficiently is chosen. Although Part I of this thesis showed that synchronization even of the simple TAG formalism leads to NP-complete recognition and that recognition of TL-MCTAG alone is NP-complete, heuristic algorithms may yet be developed that can run efficiently and in practice even the exact algorithms may run in far less than the worst case time.

I begin in Section 5.1 by introducing the use of STAG for syntax and semantics with a simple example and revisit the highly motivating example of quantifier scope discussed in Shieber and Schabes [1990], showing how use of TL-MCTAG and multiple

adjunction to model quantifiers can result in a natural prediction of scope ambiguity. From among the formalisms discussed in Part I, I begin with a focus on synchronous TL-MCTAG as a base formalism. As is apparent even from the introductory examples, two aspects of TL-MCTAG are critical to the correct predictions made by the grammars presented in this chapter: multicomponent tree sets, a tree-level domain of locality. Multicomponent tree sets in combination with multiple adjunction are used to model noun phrases, and in particular quantifiers and Wh-words, to produce the flexibility in quantifier scope ordering that we find in natural language. The tree-local domain of locality contributes to correct predictions about a wide variety of locality constraints and movement restrictions including restrictions on scoping out of the finite clause, limits on extraction such as the Complex NP constraint, and topicalization and covert movement.

Certain phenomena and constructions cannot be modeled with TL-MCTAG alone as the base formalism. We explore these cases and how they can be handled using limited delay V-TAG (see Chapter 3) in Chapter 6.

5.1 Background

The underlying idea in using synchronous TAG to model the syntax-semantics interface is that each lexical item is comprised of one or more trees that model its syntactic structure paired with one or more trees modeling its semantic structure. Although the structure of the syntactic pieces and the semantic pieces may differ substantially, it is intuitively clear that there are relationships between internal locations with the syntax and semantics of each of the lexical items. We make these

connections explicit by linking them in the formalism. When two lexical items combine, the syntax of one item connects to the syntax of the other at the same time the semantics of one item connects to the semantics of the other. The way in which the items combine is governed by the linked locations: a syntax operation occurring at a given link must occur in tandem with a semantic operation at the same link.

The syntactic trees in the elementary tree sets will likely appear familiar to those commonly seen in TAG syntactic analyses. The internal nodes of these trees are labeled with syntactic categories. The frontier nodes are either terminal nodes labeled with words from the lexicon or nonterminal nodes marked for use by one of the TAG operations and labeled with with syntactic categories. The semantic trees are less familiar but are similar in concept. The lexical content of a semantic tree is a λ-term representing the meaning of the lexical item. The internal nodes of the semantic trees are marked with semantic types. The frontier nodes are either terminal nodes labeled with meanings of words from lexicon or other elements of the λ-term such as λs or variables, or nonterminal nodes marked for use by a TAG operation and labeled with semantic types.

We proceed in this chapter to build up a grammar beginning with the most basic grammatical constructions, such as predication and modification, and then proceeding through increasingly complex constructions (quantification, wh-movement, embedding verbs, restrictions on movement out of islands and clauses, and binding theory) showing how they can be analyzed using synchronous TL-MCTAG. In the process we demonstrate the elegance with which TL-MCTAG handles (and sometimes predicts) the constraints on locality and movement that often pose challenges even

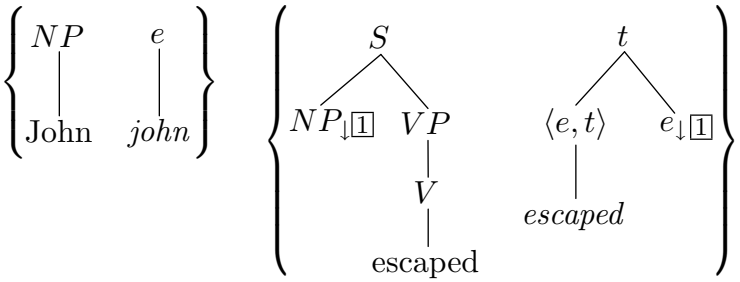


Figure 5.1: A STAG grammar for the sentence "John escaped."

for systems that produce syntax alone and certainly for systems that seek to provide semantics as well.

5.2 Predication

We begin with a simple example of predication:

(1) John escaped.

The elementary tree pairs used to derive sentence (1) are given in Figure 5.1. The syntax of *john* is a noun phrase nonterminal node dominating the single terminal node containing the lexical content. The semantics of *john* is a nonterminal node labeled with type e for entity and dominating the representation of the meaning of *john*. The syntax for the verb *escaped* demonstrates the expected structure for an intransitive verb with the location for the subject position marked for substitution of a noun-phrase in order to form a complete sentence. The semantics similarly shows *escaped* to be a function of type $\langle e, t\rangle$ (entity to truth value) applied to an entity that will be received by substitution in order to produce a complete sentence with a truth

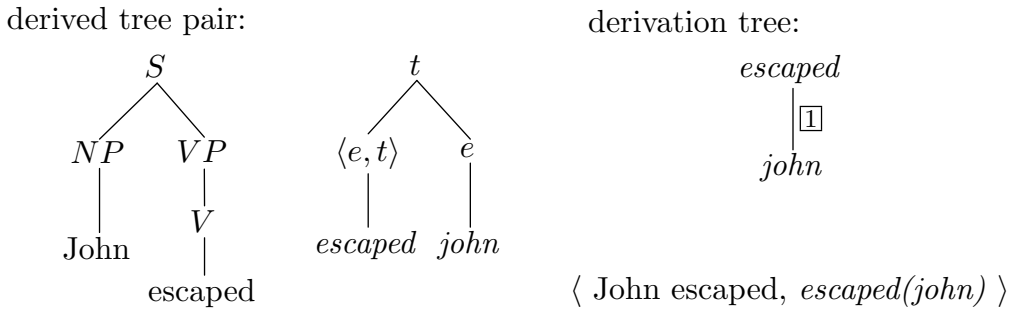


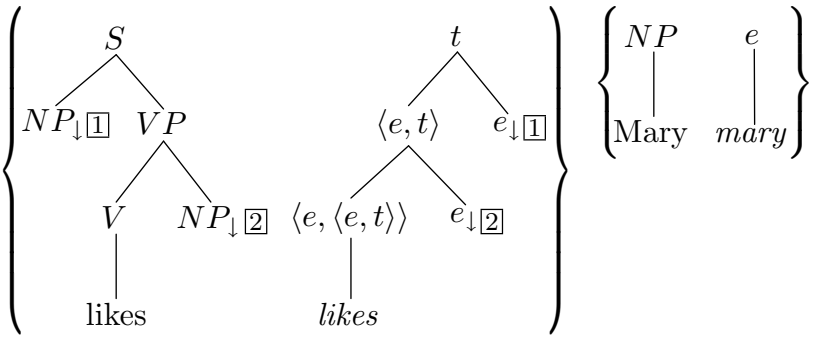
Figure 5.2: The derived tree pair and derivation tree for the sentence "John escaped."

value as its semantic type.

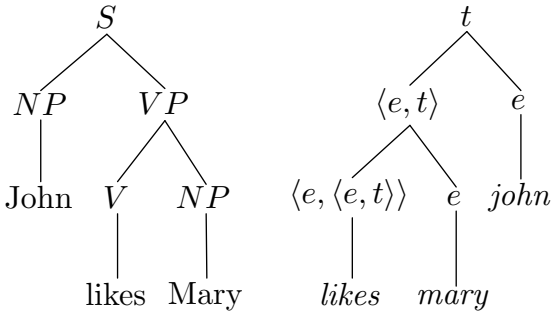
The derivation of sentence (1) is accomplished by a single TAG substitution operation in which *john* substitutes into *escaped* at link 1 in the *escaped* tree. The resulting trees as well as the derivation tree are given in Figure 5.2. The sentence can be read off the resulting derived syntactic tree by reading the terminal symbols from left to right. The λ-term for the meaning of the sentence can be read off the semantic tree by treating the leftmost terminal symbol on a branch as a functor and the remaining siblings as its arguments. Standard β-reduction can be used to reduce the λ-term if necessary. In the case of sentence (1), the lexical content of the derived tree is "John escaped" and the semantic content is *escaped(john)*.

The derivation tree fully specifies the operations necessary to reproduce this derivation. In the derivation tree *john* is a child of *escaped* because it substitutes into *escaped* in the derivation. The arc connecting them is marked with the link at which the operation occurred to specify the locations in the *escaped* tree set at which the substitution took place.

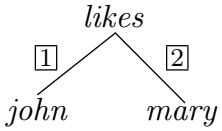
A transitive verb, such as *likes*, provides a slightly more complex example:



derived tree pair:



derivation tree:



⟨ John likes Mary, *likes(mary,john)* ⟩

Figure 5.3: Additional lexical items for an STAG grammar for the sentence "John likes Mary" as well as the derived tree pair and the derivation tree for the sentence.

(2) John likes Mary.

The additional lexical items as well as the derived tree pair and derivation tree for sentence (2) are given in Figure 5.3. In this example, the syntax of the verb offers two sites for substitution, one for the subject and one for the object. In correspondence, the semantics also contains two substitution sites for the two semantic roles. The links are numbered so that the subject in the syntax corresponds to the subject in the semantics and the object in the syntax corresponds to the patient in the semantics.

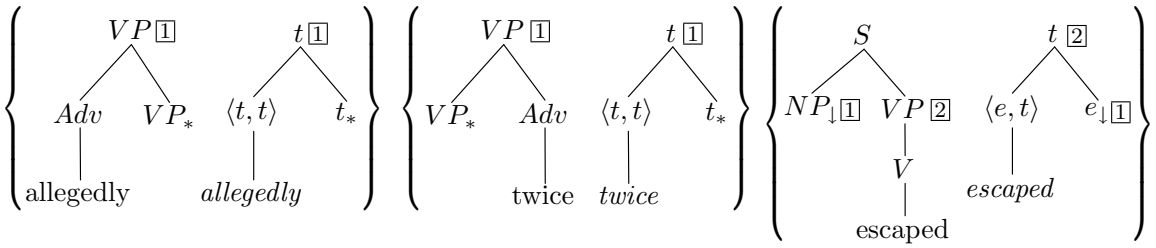


Figure 5.4: Lexical entries for the adverbs in the sentences "John allegedly escaped" and "John allegedly escaped twice" as well as an updated entry for *escaped* containing a link to permit modification.

5.3 Modification

We now augment our verb trees with additional links to permit optional modification so that we can derive sentences such as:

(3) John allegedly escaped.

(4) John allegedly escaped twice.

The adjunction operation in TAG allows for very natural modeling of modification such as in the above examples. In our grammar, this analysis is embodied in the elementary tree sets for adverbs such as *allegedly* and *twice*, which appear as auxiliary trees. These trees as well as the verb tree for *escaped* updated with a link to permit modification are given in Figure 5.4. The analysis of sentence (3) is straighforward and is given in Figure 5.5. The adverb adjoins at link 2, which is the verb phrase node in the syntax but which results in the adverb modifying the entire proposition in the semantics.

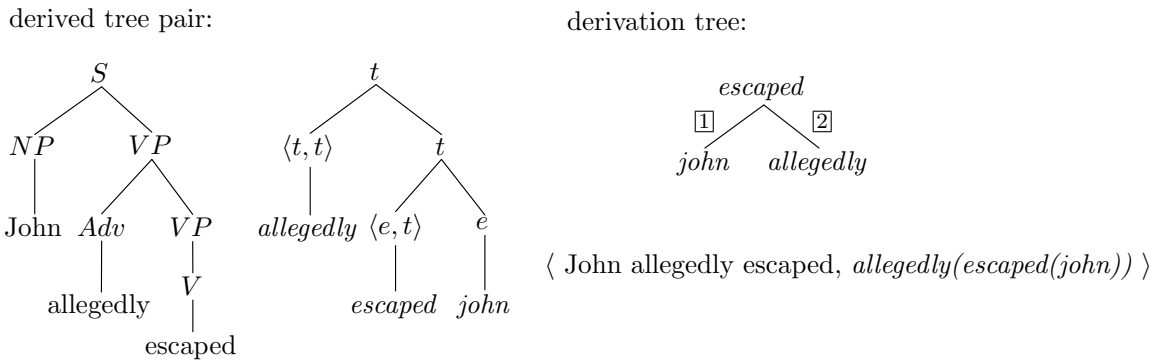


Figure 5.5: Derived tree pair and derivation tree for the sentence "John allegedly escaped."

Sentence (4) provides an interesting example of the semantic ambiguity that can arise from multiple modifications. This sentence produces two readings, one in which the proposition that is alleged is that John escaped twice and one in which has twice been the case that John allegedly escaped (whether or not he actually escaped in either attempt). The usual method for handling multiple modification in TAG syntax is to have one modifier adjoin to the other modifier so as to avoid use of multiple adjunction. Now that we have incorporated semantics, the choice of which adverb adjoins to the other adverb will also determine the order in which they take scope in the semantics. If we adjoin *allegedly* into *twice* we will produce the reading where *allegedly* outscopes *twice*: *allegedly(twice(escaped(john)))*. If we adjoin *twice* into *allegedly* we will produce the reading in which *twice* outscopes *allegedly*: *twice(allegedly(escaped(john)))*. Although the resulting sentence is the same, these are two distinct derivations in which both the syntactic and semantic trees exhibit different structure. These two analyses are shown in Figure 5.6.

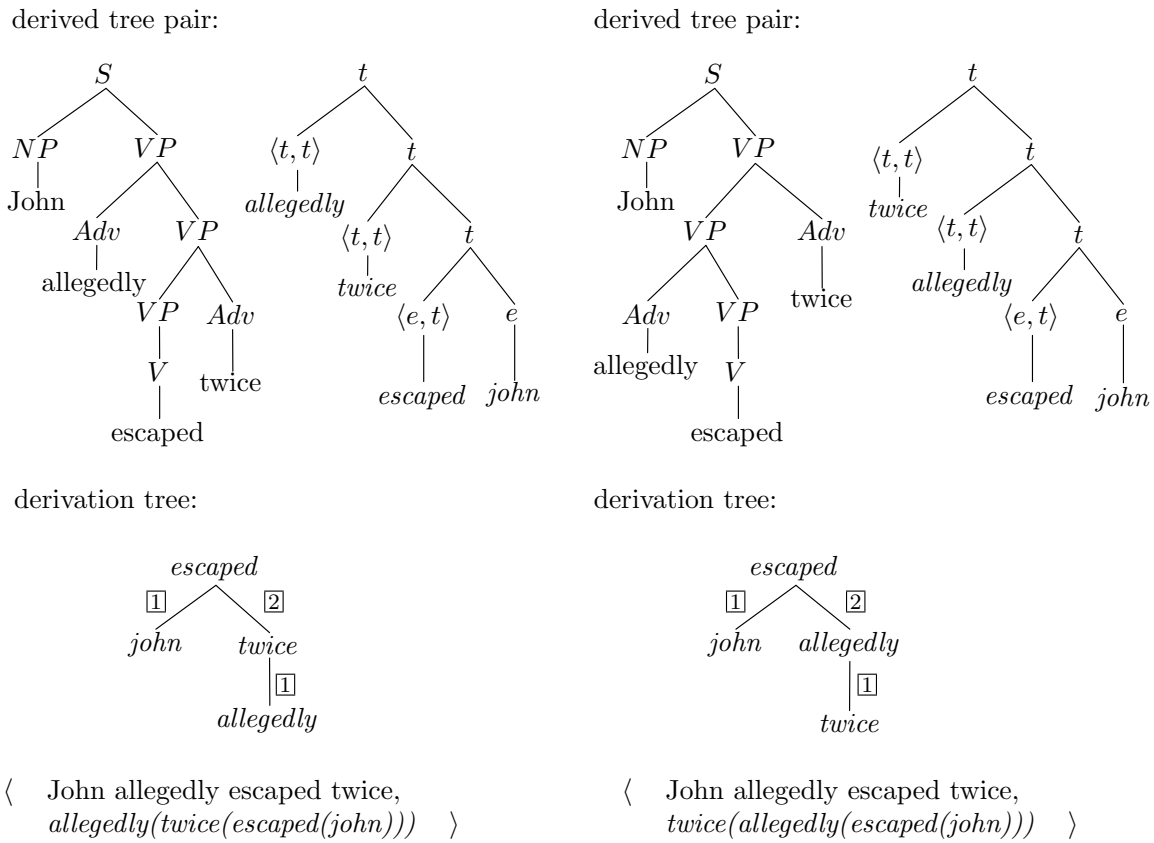


Figure 5.6: Derived tree pairs and derivation trees for the two readings of "John allegedly escaped twice." using the usual TAG method of adjoining multiple modifiers into each other before attaching to the main verb.

A third analysis of sentence (4) makes use of multiple adjunction to capture the semantic (and syntactic) ambiguity. In this analysis, both *allegedly* and *twice* are held in an equal relationship with respect to the verb and the scope ambiguity between them is captured in the derivation, which may be resolved with either one taking the wider scope. The derived trees and resulting strings and interpretations are the same as given in Figure 5.6 but they are produced from a single derivation tree in which the order of attachment at link 2 in the *escaped* tree is underspecified. This derivation

derivation tree:

⟨ John allegedly escaped twice, *allegedly(twice(escaped(john)))* ⟩
⟨ John allegedly escaped twice, *twice(allegedly(escaped(john)))* ⟩

Figure 5.7: Derivation tree and derived syntax and semantic pairs for the sentence "John allegedly escaped twice" using an analysis where the adverbs multiply adjoin into the main verb.

tree is given in Figure 5.7.

Although there is no ambiguity in the derived string, the ambiguity in the derived trees exists in the syntax as well as the semantics because both adverbs adjoin at the verb phrase node. This naturally raises the question whether the disambiguation order in the syntax must match the disambiguation order in the semantics. In our example, it is not possible to tell. However, in cases in which the adverbs appear on the same side of the verb in the derived syntax, this ordering between the multiply adjoined elements on the syntactic side appears to govern the scope ordering on the semantic side as well as shown by the contrast between the following sentences:

(5) John almost heroically escaped.

(6) John heroically almost escaped.

These two sentences have subtly different meanings that correspond to the different scope orderings of *almost* and *heroic*. This is a specific example of a widely accepted linguistic generalization that scope taking elements take scope in the order they appear along the VP spine in the syntax.

Although the same derived trees are produced for sentence (4) ("John allegedly escaped twice") by the two methods of analysis, other examples offer reasons to prefer the multiple adjunction analysis. Consider the phrases:

(7) roasted red pepper

(8) baked red pepper

In the analysis in which *roasted* or *baked* must adjoin into *red* before the two adjoin to *pepper* we break the direct relationship between the first adverb and the noun it modifies. However, as evidenced by this example, for applications in which one is trying to predict the likelihood of the occurrence of the word *pepper* in the context, this relationship is critical. The occurrence of *pepper* is much greater in the first phrase than the second because it is typical to refer to the process of cooking peppers in an oven as roasting rather than baking.

5.4 Basic Quantification

The semantics of quantification can be modeled easily using synchronous TL-MCTAG by augmenting our analysis of noun phrases to account for their potential to act as scope-taking elements. Consider the sentence:

(9) Everyone escaped.

Figure 5.8 shows the lexical items necessary to analyze sentence (9). The elementary tree set for *every* consists of two syntax and two semantic trees. The semantic

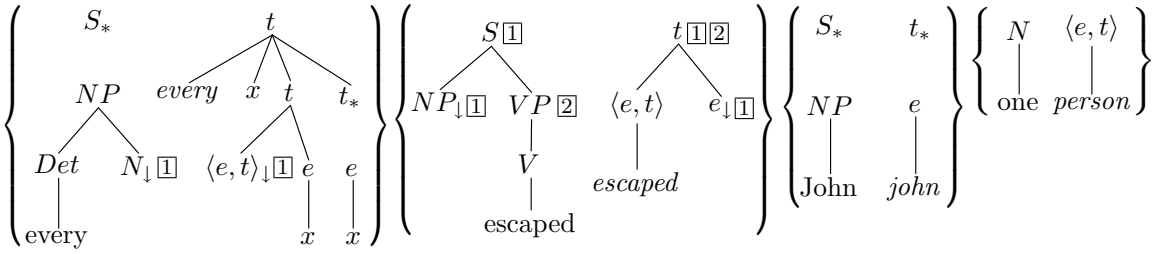


Figure 5.8: The elementary tree pairs for the sentence "Everyone escaped".

tree encodes a four-part quantifier analysis. The top tree is an auxiliary tree giving the quantifier itself, its variable, and its restriction. This tree adjoins into the nuclear scope of the quantifier. The bottom tree is an initial tree containing just the bound variable that substitutes into the nuclear scope of the quantifier. In the syntax the top tree is an auxiliary tree consisting of a single node so that when it adjoins it has no effect on the structure of the derived tree. For now, this tree is included in the tree set for symmetry with the make up of the semantic trees of the tree set, but later in this chapter its inclusion will be motivated by the analysis of other noun phrases such as Wh words and other configurations such as topicalization. The bottom tree in the syntax is the usual noun phrase tree that one expects in noun phrase syntax. In order for quantifiers to adjoin they require links that have positions for both trees in the syntax and semantics.[1] As a result, the links intended for use by noun phrases are updated to have two locations in the syntax and two locations in the semantics.

[1]The multicomponent approach to quantifiers in STAG was first suggested by Shieber and Schabes [1990] under the rewriting definition of STAG derivation where the order of rewriting produced the scope ambiguity. Williford [1993] explored the use of multiple adjunction to achieve scope ambiguity. The multicomponent quantifier approach followed by Joshi et al. [2003] is syntactically based. In their work, a single-node auxiliary tree is used for the scope part of the syntax in order to get the desired relationship between the quantifier and the quantified expression in features threaded through the derivation tree and hence in the semantics. The additional node in the syntax is not needed for this purpose because the semantics are modeled explicitly. However, as noted, its inclusion will be independently motivated.

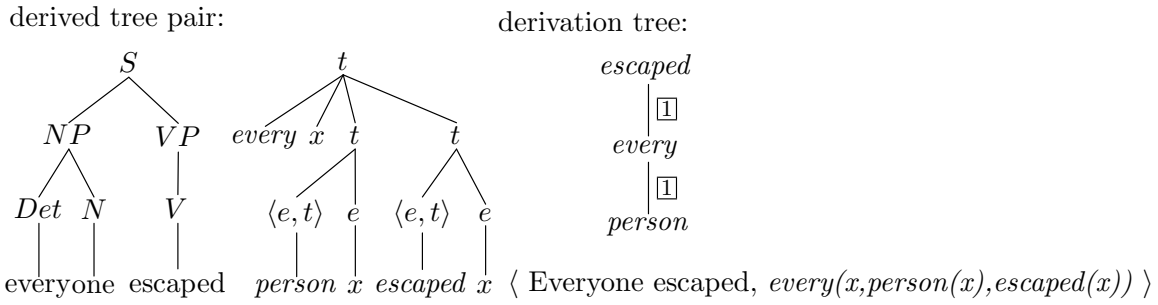


Figure 5.9: Derived tree pair and derivation tree for the sentence "Everyone escaped".

This is shown in the lexical entry for *escaped* in Figure 5.8. For uniformity the lexical entries for non scope-taking noun phrases such as *john* are also updated so that they may use these expanded links. As with the vestigial tree in the syntax of quantifiers, these vestigial trees in non-quantificational noun phrases will be independently motivated later in this chapter.

To derive sentence (9), *every* adjoins into *escaped* at link 1. The node labels of the trees determine which trees adjoin at which locations of the link.[2] This produces the derived tree pair and derivation tree shown in Figure 5.9.

The following sentence provides a more interesting and complex example:

(10) Everyone likes someone.

$every(x, person(x), some(z, person(z), like(x, z)))$

$some(z, person(z), every(x, person(x), like(x, z)))$

[2]Although in an implementation we would use the vector definition and index the trees in the elementary tree vectors, we omit indices in the elementary structures for simplicity of presentation in this chapter because the intended adjunction sites are clear from the node labels and the implicit dominance relations within elementary tree sets.

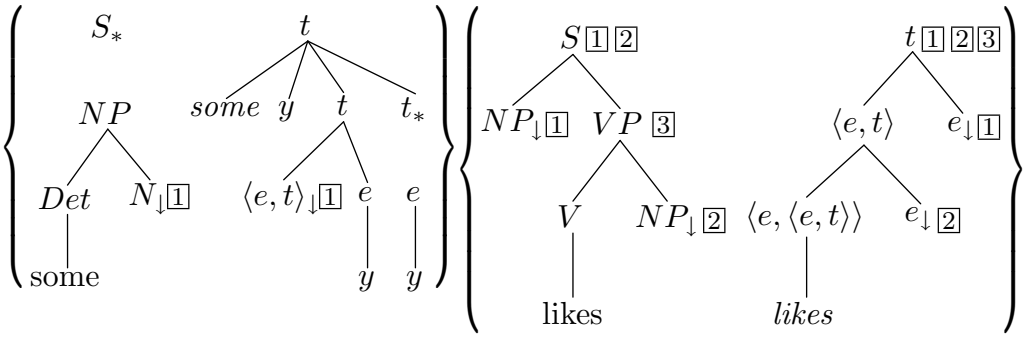


Figure 5.10: Additional elementary tree pairs for the sentence "Everyone likes someone".

For sentence (10), we would like to generate a scope-neutral semantic representation that allows both the reading where *some* takes scope over *every* and the reading where *every* takes scope over *some*. Similar to the use of multiple adjunction to model modifiers, we propose a solution in which a derivation tree with multiple adjunction nondeterministically determines multiple derived trees each manifesting explicit scope [Schabes and Shieber, 1993]; the derivation tree itself is therefore the scope neutral representation.

This results in the derivation tree and derived trees shown in Figure 5.11 for sentence (10). Note that the resulting derivation tree necessarily incorporates multiple adjunction. The scope parts of both *every* and *some* attach at the root of the semantic tree of *likes*. This induces ambiguity: the derivation is ambiguous as to which quantifier scopes higher than the other. This ambiguity in the derivation tree thus models the set of valid scopings for the sentence. In essence, this method uses multiple adjunction to model scope-neutrality.

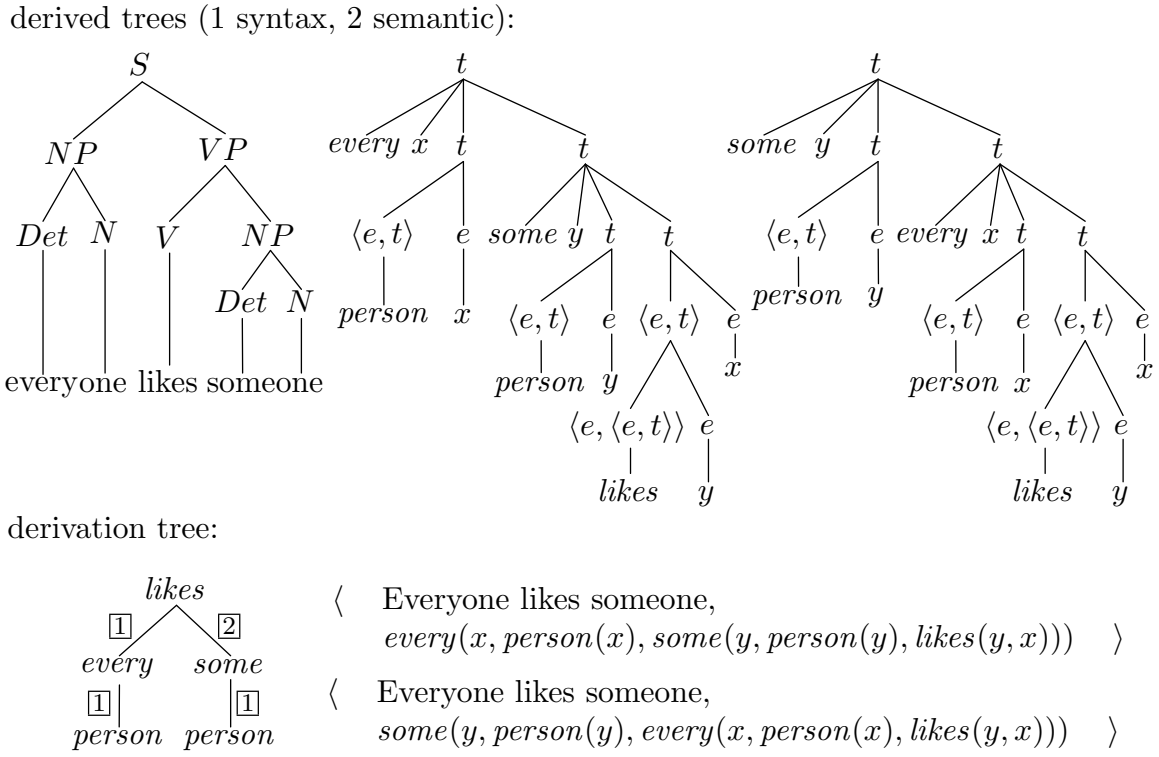


Figure 5.11: The derivation tree and derived syntactic and semantic trees for the sentence "Everyone likes someone". Note that the derivation tree is a scope neutral representation: depending on whether *every* or *some* adjoins higher, different semantic derived trees and scope orderings are obtained.

5.5 Wh Questions

In TAG syntax, Wh questions are typically formed using special alternative verb trees that have a location for the Wh word and lack a location for one of the usual noun phrase arguments. The inclusion of semantics in this system and the structure already adopted for noun phrases suggests a more semantically plausible method of analyzing Wh words that does not require the addition of alternative verb trees.

Rather than marking the coindexation of the Wh word and its trace on the verb tree (or failing to mark it at all), Wh words are treated much like quantifiers with

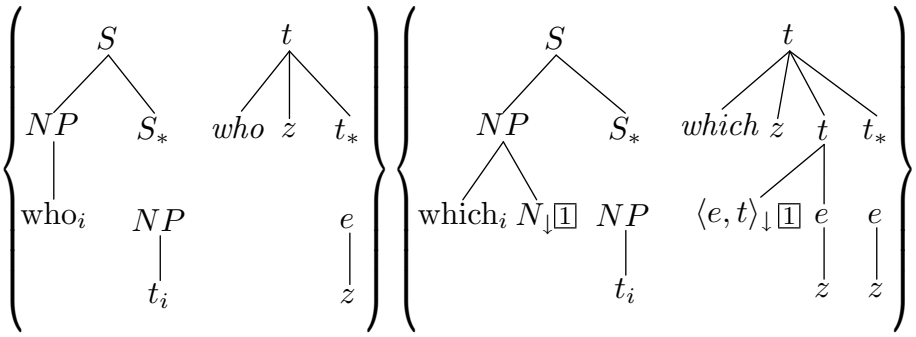


Figure 5.12: The lexical entry for *who*.

tree sets in which both the syntax and semantics of the Wh word contain two trees. In the syntax, one tree contains the lexical content of the Wh word and the other tree contains the coindexed trace in a noun phrase that can fill one of the argument positions of the verb to which the Wh word attaches. In the semantics, a Wh word consists of a auxiliary tree that includes the wh word, and its variable and adjoins into its scope and an initial tree that contains only the bound variable and substitutes into the scope of the Wh question. Structurally, then, Wh word lexical entries are very similar to those of other noun phrases: they consist of two syntactic trees (an auxiliary tree with root and foot labeled with S and an initial tree with root labeled with NP) and two semantic trees (an auxiliary tree with root and foot labeled with t and an initial tree with root labeled with e). The form and use of Wh words is demonstrated by the analysis of the following sentences using the lexical entries for *who* and *which* given in Figure 5.12:

(11) Who escaped? (Who$_i$ escaped t_i?)

(12) Which person escaped? ([Which person]$_i$ escaped t_i?)

For both sentence (11) and sentence (12), the Wh word adjoins to the main verb, both filling the argument position and forming the Wh question with the Wh word fronted. The resulting derived tree pairs and derivation trees are given in Figure 5.13.

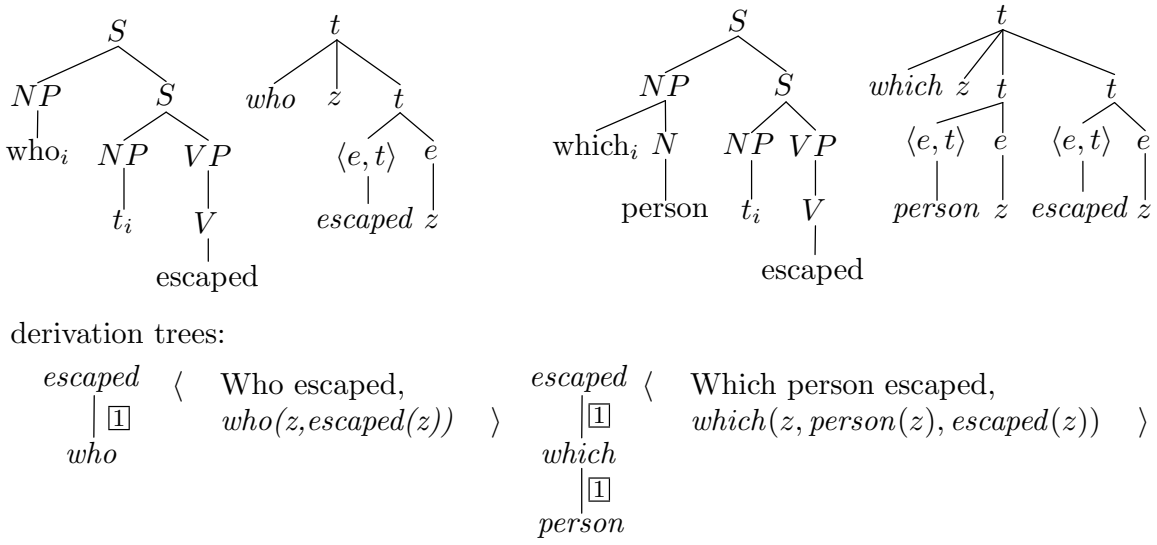


Figure 5.13: The derived tree pairs and derivation trees for the sentences "Who escaped?" and "Which person escaped?".

It is necessary to add something additional to prevent the formation of multiple wh-questions within the same finite clause as exemplified by the following sentence:

(13) *Who does who like? (Who$_i$ does who$_j$ t_j like t_i?)

We use finite features in the semantic tree to enforce this restriction. We introduce a feature WH that has values drawn from the set {+,-}. An example of how this feature is used to rule out Sentence (14) is given in Figure 5.14.

The interaction of quantifiers and Wh words falls out naturally from the lexical items already introduced, as exemplified by the following sentence:

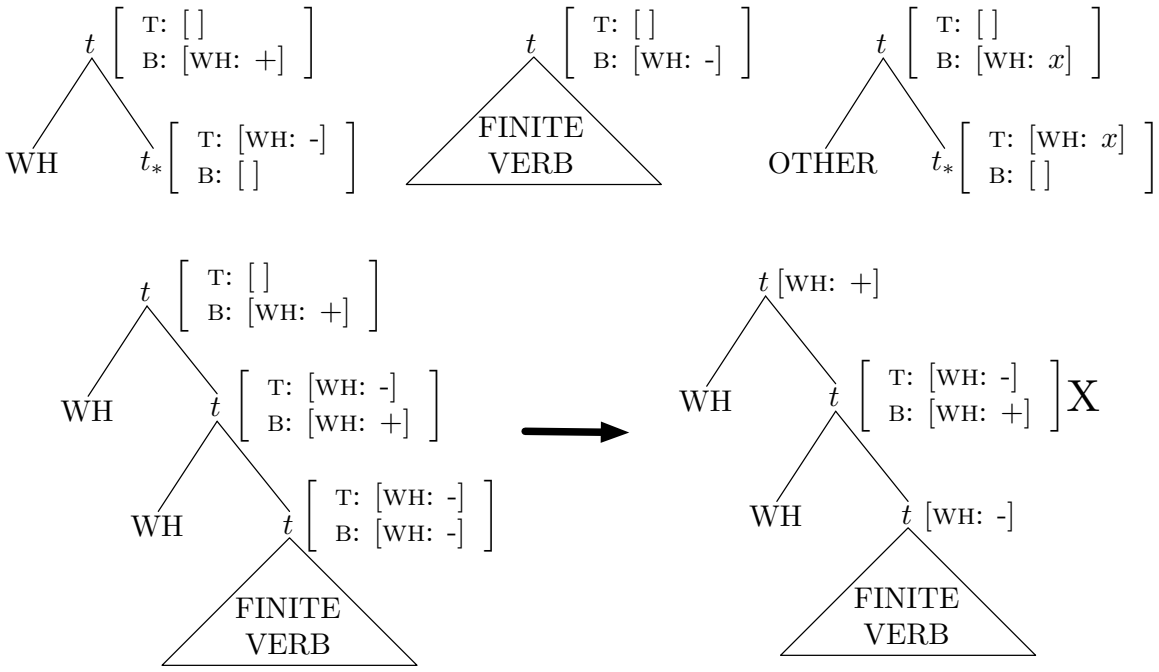


Figure 5.14: Schematic representations of lexical items for Wh words and finite verbs and other lexical items that adjoin at t are given along with a diagram demonstrating the invalidity of the sentence "Who$_i$ does who$_j$ t_j like t_i?". Note that other lexical items that adjoining at t do not affect the number of Wh words permitted to adjoin to a single finite verb because they simply pass along the WH feature. (Analysis of do-support is omitted for simplicity.)

(14) Who does everyone like? (Who$_i$ does everyone like t_i?)

$who(z, every(x, person(x), like(z, x)))$

$every(x, person(x), who(z, like(z, x)))$

The derived trees and derivation tree for sentence (14) are given in Figure 5.15. As with sentences with multiple quantifiers, the scope ambiguity between the Wh word and the quantifier arises directly from the multiple adjunction of their scope parts at

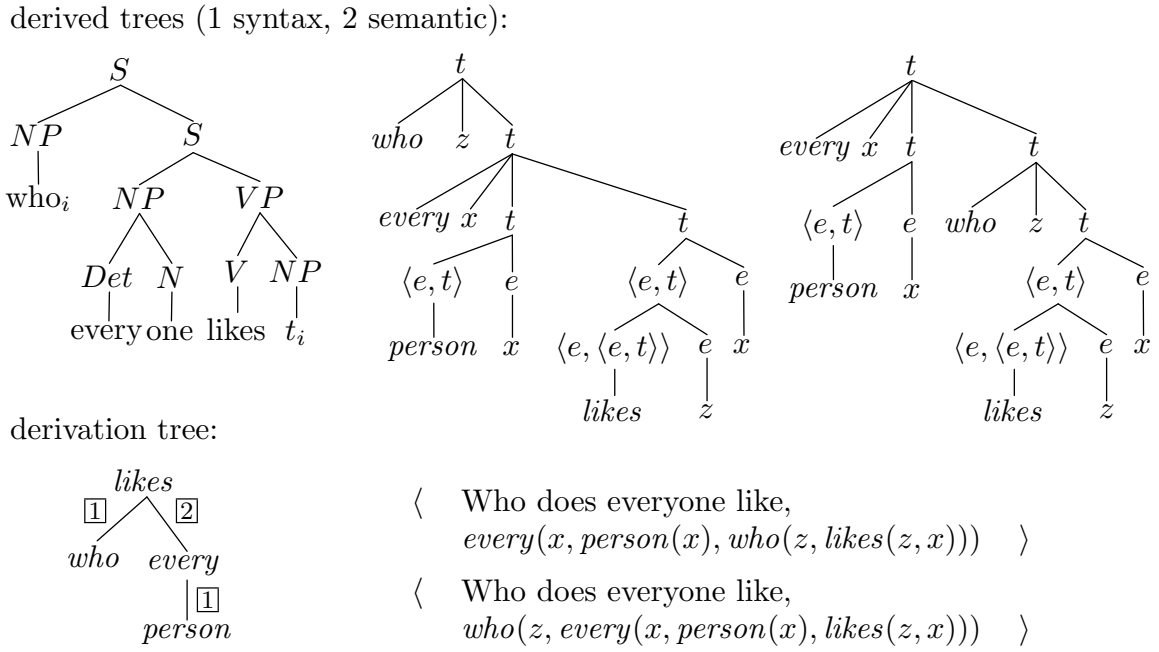


Figure 5.15: Derived trees and derivation tree for the sentence "Who does everyone like?"

the root of the tree for *likes* in the semantics.

5.6 In Situ Wh Questions and Topicalization

The two-part analysis of the syntax and semantics of Wh words suggests a simple and elegant correspondence between their usual fronted location in the syntax and their unmoved location in in situ Wh questions (also called covert movement). An alternative elementary tree set is added for each Wh word that characterizes the in situ Wh question syntax. In these tree sets the lexical content of the Wh word is moved to the NP-rooted tree rather than appearing in the S-rooted tree. The semantic trees from these alternative sets are the same as in the standard Wh word

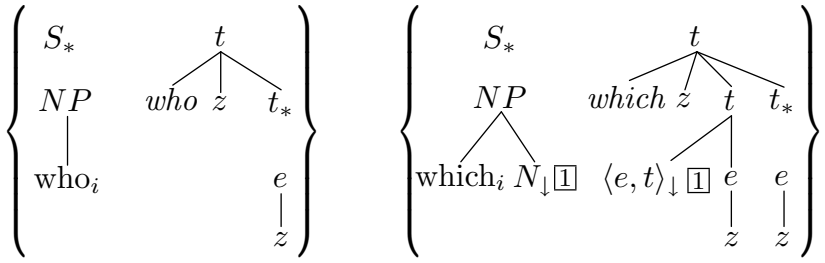


Figure 5.16: Alternative elementary tree sets for Wh words used to model in situ Wh questions. Note that the semantics is unchanged from the standard Wh word elementary tree sets and that the syntactic trees have the standard noun phrase signature of an S-rooted auxiliary tree and and NP-rooted initial tree.

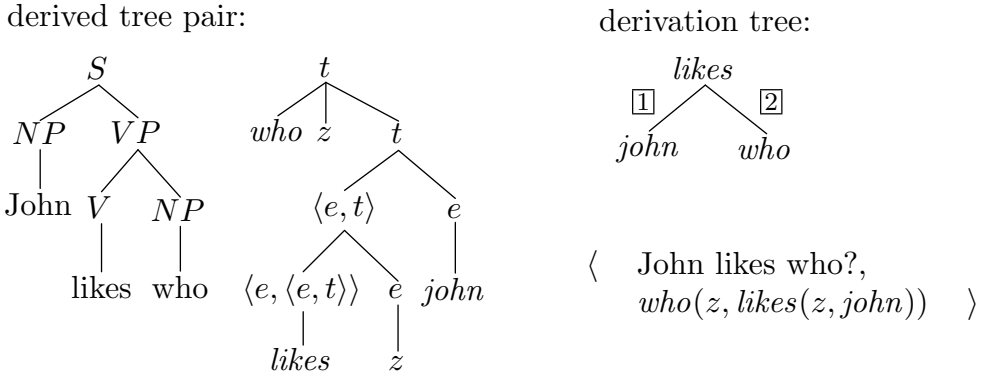


Figure 5.17: Derived tree pair and derivation tree for the sentence "John likes who?"

elementary tree sets and the syntactic signature—the number of trees in the set and their root and foot node labels—is the same as in the standard Wh word elementary tree sets. Using the alternative syntax tree set given in Figure 5.16 we model in-place use of wh-words as in sentence (15) while still maintaining the usual semantic analysis (Figure 5.17).

(15) John likes who?

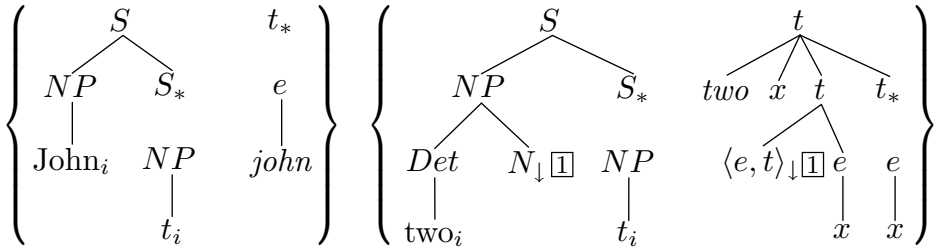


Figure 5.18: Alternative tree pairs for *John* and *two* that model topicalization. Note that the semantics are unchanged from the standard elementary tree sets.

This insight extends to an elegant analysis of topicalization as well. The vestigial S_* tree that we added to the tree set for the syntax of every non-quantificational noun phrase need not always be contentless. Just as we moved the lexical content of *who* from the S-rooted tree in its set to the NP-rooted tree to model in situ wh-words, the lexical content of noun phrases may be moved to the top tree in their sets to model topicalization. For instance, the alternative tree pair for *John* shown in Figure 5.18 provides for an analysis of sentence (16) (Figure 5.19).

(16) John$_i$, Mary likes t_i.

The analysis also works for topicalized quantificational NPs so that sentence (17) follows from the tree pair for *two* also given in Figure 5.18. The analysis sentence (17) is given in Figure 5.19.

(17) [Two books]$_i$, Mary likes t_i.

Although the structure of the semantic trees is the same in the usual elementary tree sets and the alternative elementary tree sets presented in this section, there are likely to be subtle differences in the semantics that will be represented by a difference in

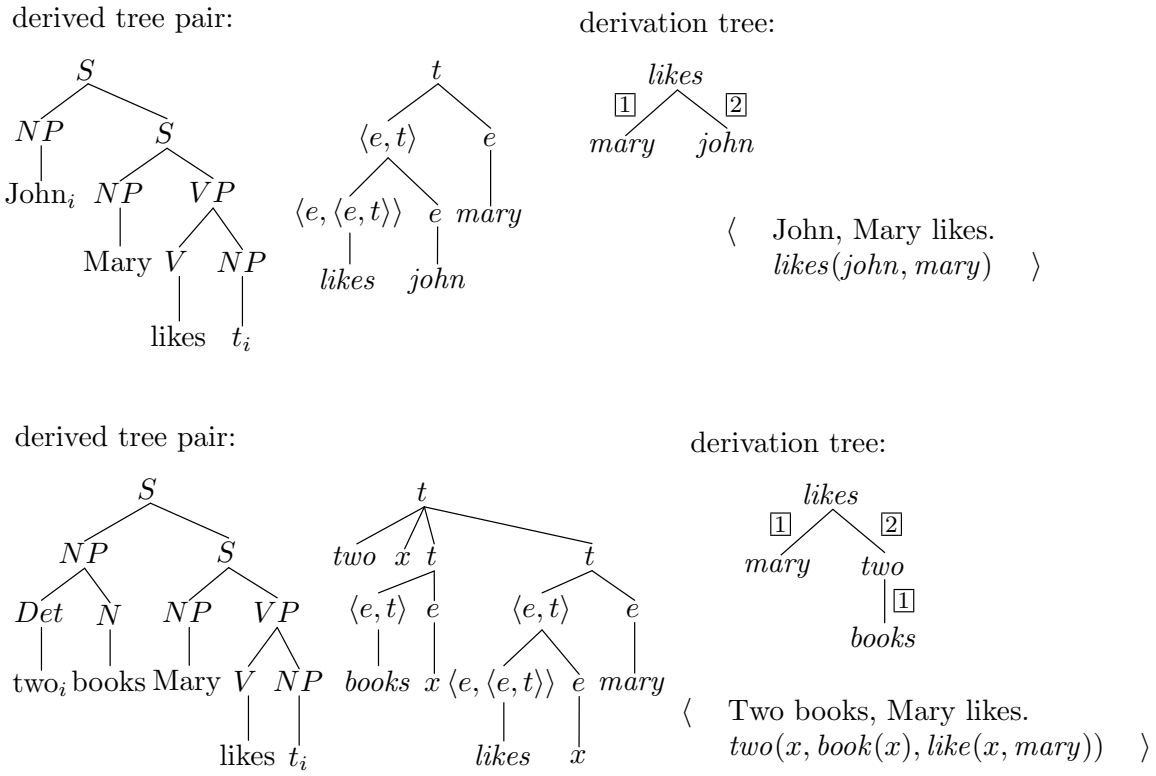


Figure 5.19: Derived tree pairs and derivation trees for the sentences "John, Mary likes." and "Two books, Mary likes."

features in the elementary tree sets. For instance, in the topicalization case a feature in the semantics may indicate a focus on the topicalized entity.

5.7 Raising, Embedding and Control Verbs

5.7.1 Raising Verbs

Structural differences in the lexical entries for embedding, raising, and control verbs can straightforwardly account for their different properties. Raising verbs, such as *seem*, do not themselves assign a semantic role to their subjects. Even though

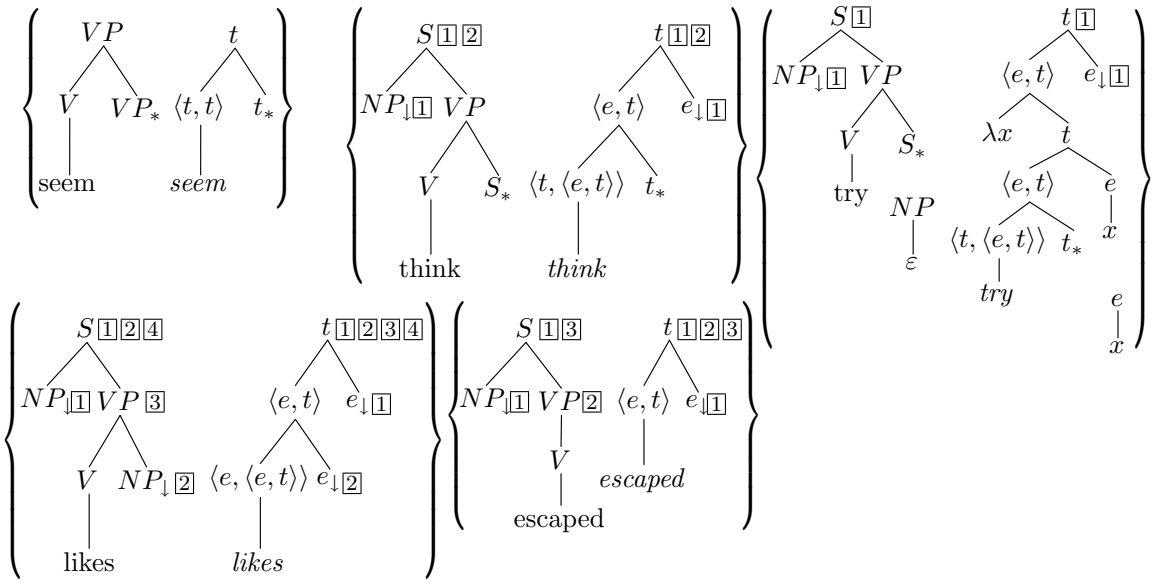


Figure 5.20: Lexical entries for example raising (*seem*), embedding (*think*), and control (*try*) verbs. The raising verb does not take its own subject but adjoins in under the subject of another verb. The embedding verb has its own subject independent from the subject of the verb it adjoins to. The control verb takes a subject and uses a bound variable to control the subject (or object) of the verb it adjoins to. An updated lexical entries for verbs *escape* and *like* are also included showing an additional link at the root of the syntax and semantics trees that can be used by embedding and control verbs.

the subject appears in the subject position of the raising verb, the semantic role is assigned by the verb in the lower clause. Consider the sentence:

(18) John seemed to escape.

For this sentence we wish to generate the semantics *seem(escape(john))*. Typically in TAG syntax, raising verbs are analyzed as verb phrase modifiers so that they do not contribute their own subject to the syntactic tree but instead insert themselves underneath the subject of the lower verb in the sentence. Following this practice, if we analyze raising verb semantics the same way that we analyze other verb phrase

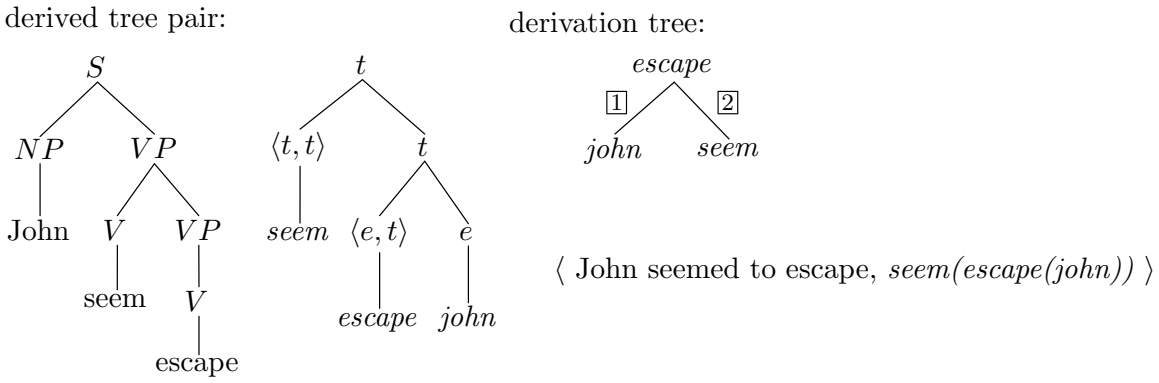


Figure 5.21: Derived tree pair and derivation tree for the sentence "John seemed to escape". Aspects of the derivation that are accomplished with syntactic features such as verb form and tense are omitted for simplicity.

modifiers in our lexicon such as adverbs, we produce the desired result. That is, we analyze the semantics of raising verbs as auxiliary trees that adjoin at the level of the proposition in the semantics. The lexical entry for *seem* demonstrating this analysis is given in Figure 5.20. The derived trees and derivation tree for sentence (18) are given in Figure 5.21.[3]

5.7.2 Embedding Verbs

Embedding verbs, such as *think* and *say*, take complete sentences as their complements and also have a syntactic subject to which they assign a semantic role. This exemplified by the following sentence:

[3]This analysis makes the prediction that quantifiers and raising verbs should scope freely with respect to each other. This is clearly true in sentences such as the following in which both a *de dicto* and *de re* reading are available:

(19) Someone seemed to escape.

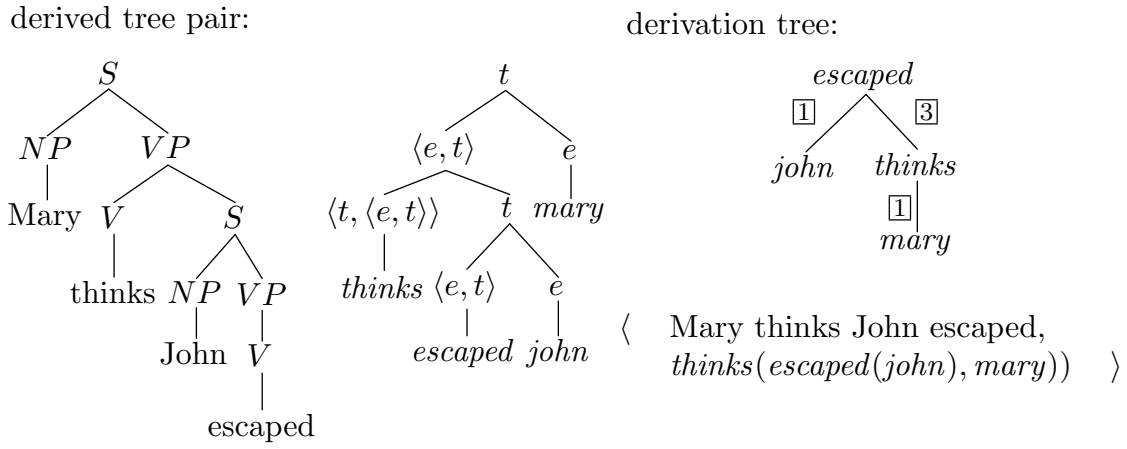


Figure 5.22: Derived tree pair and derivation tree for the sentence "Mary thinks John escaped".

(20) Mary thinks John escaped.

It is standard in TAG syntax to analyze embedding verbs as auxiliary trees that take their complements by adjoining into them. Straightforward extension of this analysis to include semantics (Figure 5.20) produces a satisfactory analysis for sentence (20), as shown by the derived trees and derivation tree given in Figure 5.22. The motivation for this analysis of embedding verbs as opposed to one in which the verb takes its complement by substitution arises from the acceptability of unbounded long-distance movement of Wh words (and pied-piped accompaniments to them) beyond the boundary of the finite clause. This is exemplified by the following sentence:

(21) Who$_i$ does Bill think Paul says John likes t_i?

As shown in the derivation tree in Figure 5.23, in sentence (21), the Wh word adjoins to the main verb *likes*. The embedding verbs adjoin to each other in a chain and

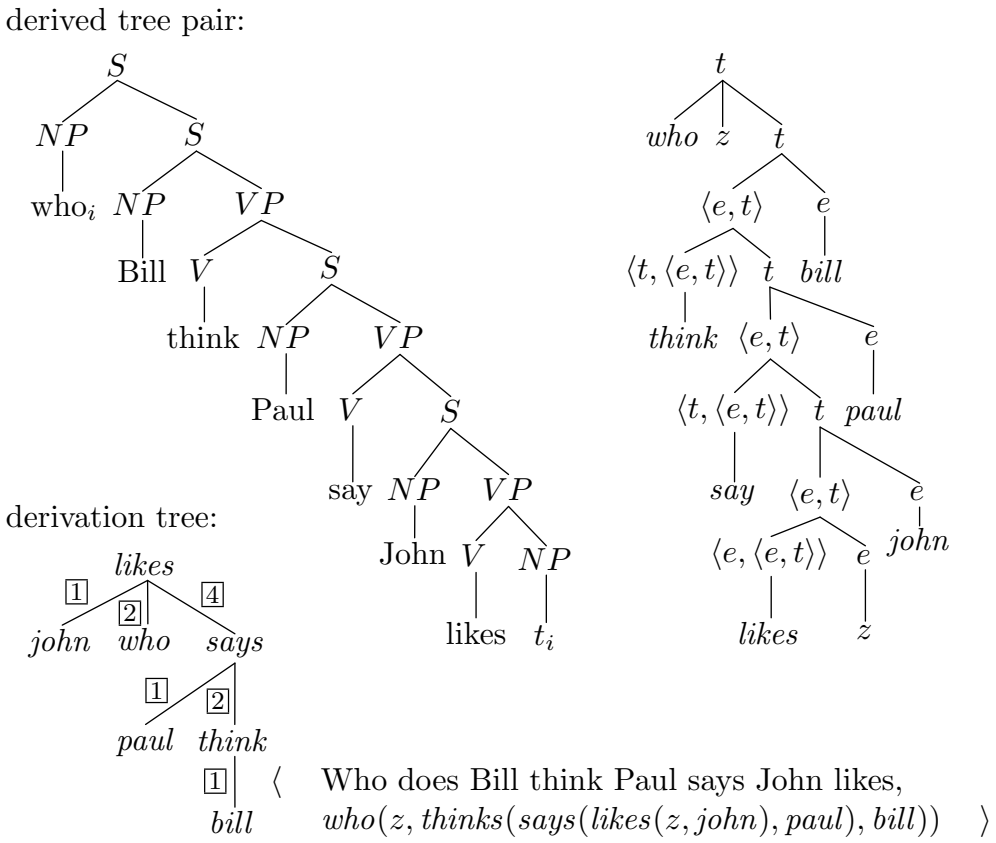


Figure 5.23: Derived tree pair and derivation tree for the sentence "Who does Bill think Paul says John likes?". The tree pair for *says* is omitted because it are isomorphic the one for *think*. Similarly, we leave lexical items for *Bill* and *Paul* implicit.

ultimately adjoin to the main verb. Because the embedding verbs adjoin to the main verb, an arbitrary distance can arise between the Wh word and its trace. Syntactically, this analysis works well but Kallmeyer and Romero [2004] highlight this case as difficult for TAG semantics because the derivation tree appears to lack the necessary direct relationships: there is no link in the derivation tree between *who* and *thinks* or between *thinks* and *likes*, but in the desired semantics *who* takes scope over the *thinks*

proposition and the *likes* proposition is an argument to *thinks*. In our analysis, however, the semantics follows quite naturally because *who* and the chain of embedding verbs multiply adjoin at the level of the proposition of the main verb.

The lexical ordering of the Wh word with the embedding verbs in the syntax (where there is multiple adjunction at the sentence level) dictates as desired that the Wh word should outscope the embedding verbs in the semantics. However, this alone is not necessarily sufficient to enforce the requirement that Wh words outscope embedding verbs. For instance, in cases in which the Wh word is not fronted (as in covert movement) there is no lexical information in the syntax that can be used to dictate the relative scope of the Wh word and the embedding verbs.

A similar problem is posed by the relationship between quantifiers and embedding verbs. That quantifiers cannot scope out of their clause is modeled directly by the locality constraints of TL-MCTAG and the design of the lexicon. Due to the tree locality constraint, when the quantifier adjoins to a main verb or other lexical element, the scope of the quantifier is limited to the highest t node within the tree to which it adjoins. However, this picture is complicated by embedding verbs. Because embedding verbs such as *thinks* multiply adjoin with quantifiers in the semantics, it possible for a quantifier to scope over an embedding verb, a result that contradicts widely-accepted syntactic theory.

Sentence (22) illustrates the problem:

(22) Bill thinks John likes two books.

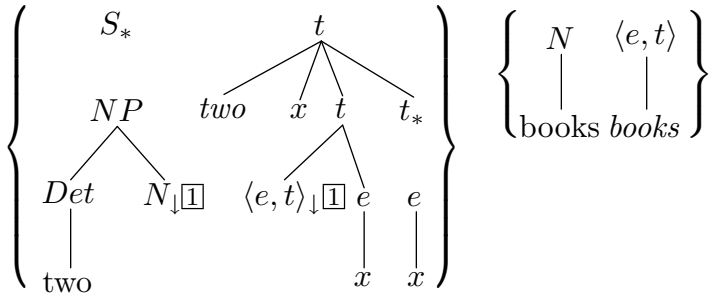


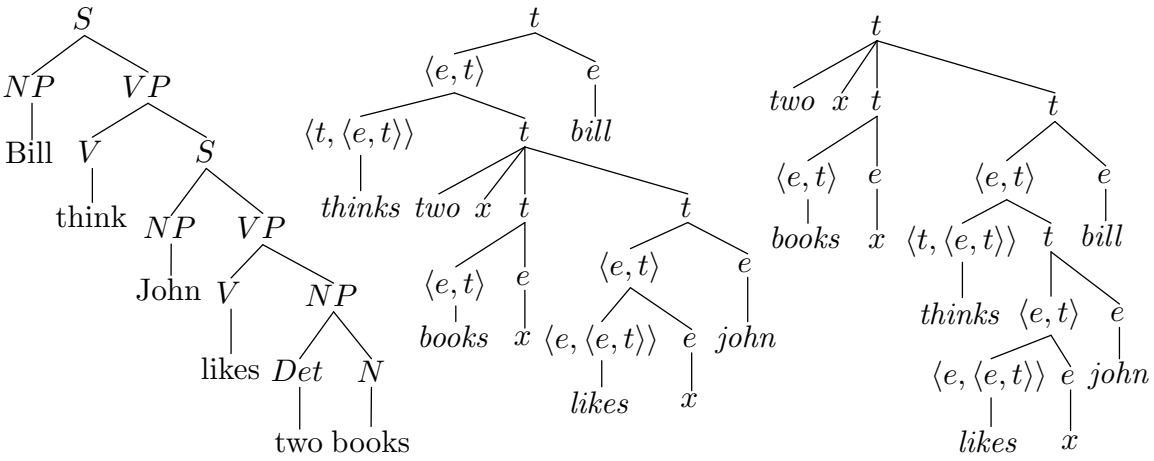
Figure 5.24: Lexical entries for *two* and *books*.

Additional lexical items are given in Figure 5.24 and the derivation tree and derived semantics are given in Figure 5.25. The potential for multiple derivations for this sentence is made possible by the multiple adjunction of *thinks* and *two* at the same node in the semantics with nothing to specify which of these two elements should adjoin higher. In this case, the embedding verb must somehow be constrained to adjoin higher than the quantifier.

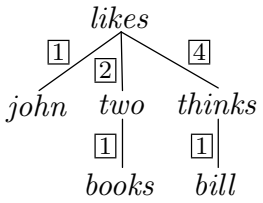
If this were the only complication it could be fixed simply by separating the t node into two nodes and enforcing that the embedding verbs adjoin at the higher node and the quantifiers at the lower node. However, as shown above, Wh words and quantifiers can scope freely with respect to each other when no embedding verb intervenes between them. If embedding verbs were forced to adjoin at a higher node than quantifier scopes, this desirable outcome would be lost. Quantifiers must scope under embedding verbs and Wh words must scope over them but Wh words and quantifiers must remain free to scope in either order when embedding verbs are not present. This state of affairs is exemplified by sentence (23).

(23) Who$_i$ does Bill think everyone likes t_i?

derived trees (1 syntax, 2 semantic):



derivation tree:

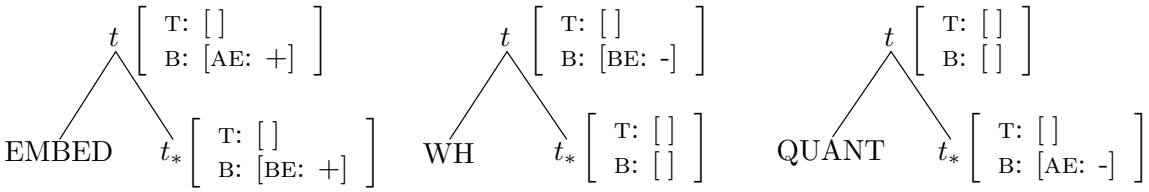


⟨ Bill thinks John likes two books.
thinks(*two*(*x*, *book*(*x*), *likes*(*x*, *john*)), *bill*) ⟩

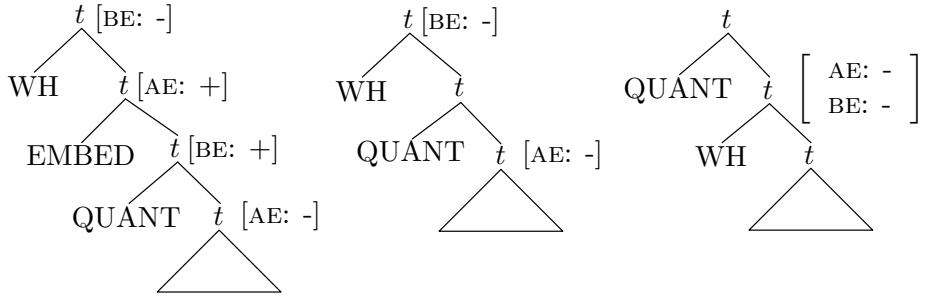
⟨ Bill thinks John likes two books.
two(*x*, *book*(*x*), *thinks*(*likes*(*x*, *john*), *bill*)) ⟩

Figure 5.25: Derived trees and derivation tree for the sentence "Bill thinks John likes two books." The derived semantics on the left is the desired semantics. The one on the right demonstrates the quantifier scoping out of the finite clause. Both semantic trees are represented by the single derivation tree shown.

Semantic features can be used enforce the valid orderings. Two features, AE and BE, which stand for above embed and below embed, respectively, are sufficient. Each of these features has two values, + and -. The features are distributed as shown in Figure 5.26. As shown in Figure 5.26, the given distribution of features prevents the quantifier from scoping above the embedding verb and the Wh word from scoping below the embedding verb but leaves the quantifier and Wh word free to scope in either order when no embedding verb is present.



valid orderings:



invalid orderings:

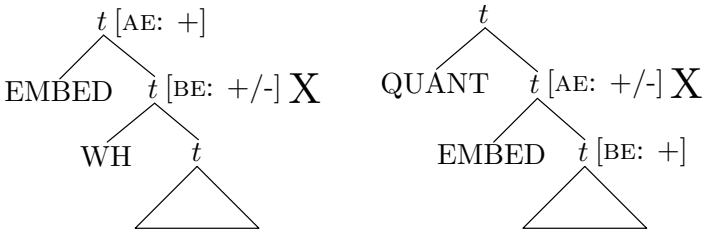


Figure 5.26: Schematic versions of the semantic parts of the lexical entries for embedding verbs, Wh words, and quantifiers that show the distribution of features used to regulate scope ordering and examples of valid and invalid orderings as determined by the features.

5.7.3 Control Verbs

Like embedding verbs, control verbs such as *try* have their own subject. Their subject, however, controls the subject (or, in some cases, object) of the lower verb as in the example sentence:

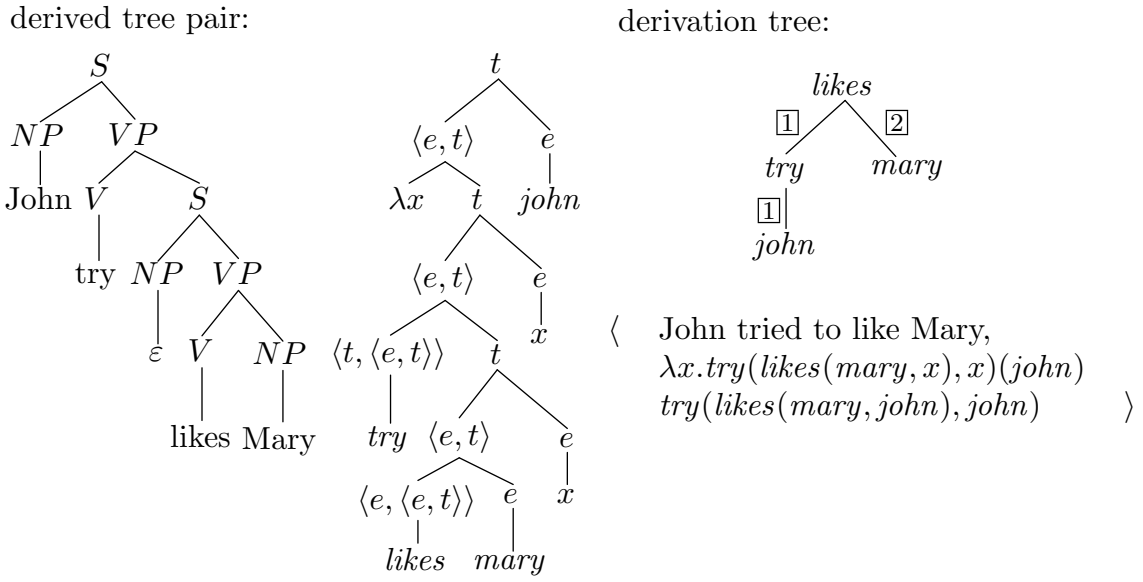


Figure 5.27: Derived trees and derivation tree for the sentence "John tried to like Mary." The lexical entry for *try* is given in Figure 5.20. The details of generating the infinitival verb form for *likes* are omitted.

(24) John tried to like Mary.

The critical aspect of a subject control verb is that the lower verb cannot have an independent subject:

(25) *John tried Bill to like Mary.

As shown in Figure 5.20, this control is accomplished using a method similar to that used for bound variables in Wh words and quantifiers. The subject of the control verb substitutes into it as expected. The control verb captures that subject in a variable and then uses it in two places, its own subject and an initial tree that substitutes into the subject (or object) position of the lower verb. Figure 5.27 demonstrates this using Sentence (24) as an example.

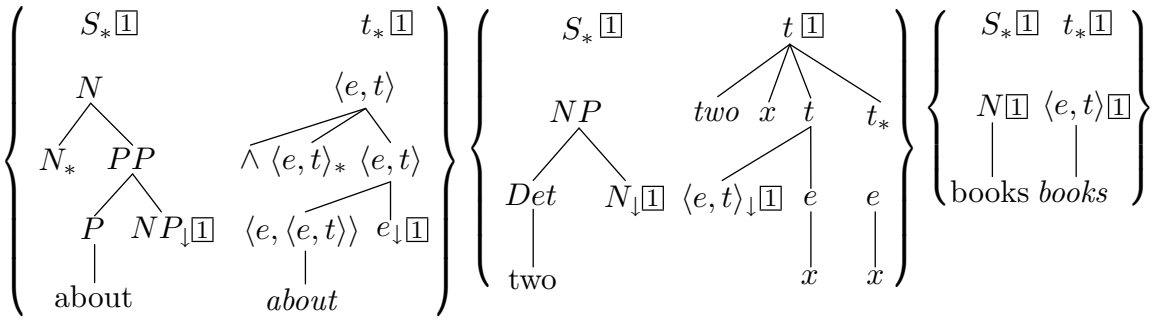


Figure 5.28: Lexical entry for the preposition *about* given as an example of a preposition modifying a noun phrase and taking a noun phrase complement. An updated lexical entry for *books* shows the addition of vestigial trees and a link. The addition of the vestigial trees gives a location where the scope parts of quantified noun phrases that adjoin into constituents headed by nouns can attach. An updated lexical entry for *two* is also given to show the updated link at which nouns may adjoin.

In contrast to embedding verbs, control verbs do not mark a finite clause boundary. As a result, the linguistic generalization that dictates that quantifiers attaching below an embedding verb cannot scope above them does not apply in the case of control verbs. This means that features are not needed to prevent quantifiers from outscoping them. However, it gives rise to another complication. Under our current analysis, when a quantifier attaches to a control verb its scope part becomes indivisibly attached to the semantics of the control verb. This prevents the scope of a quantifier that attaches to the lower verb from intervening between them. This behavior is undesirable. We defer the treatment of this issue to Chapter 6.

5.8 Prepositional Phrases

Prepositional phrases may modify noun phrases (attaching to the right), verb phrases (attaching to the right) or sentences (attaching to the left). In addition, they

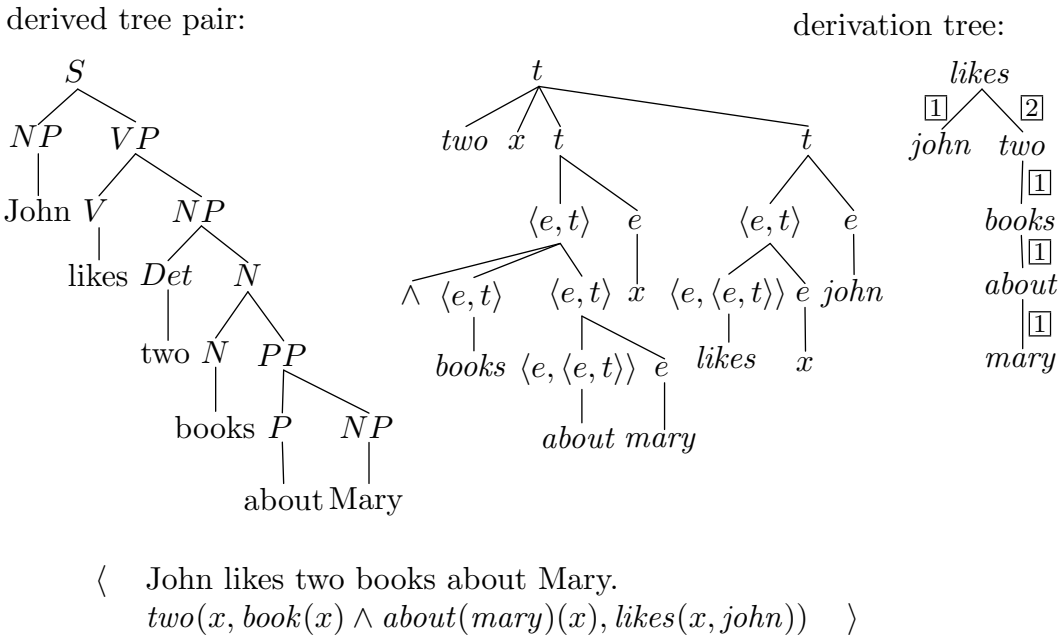


Figure 5.29: Derived tree pair and derivation tree for the sentence "John likes two books about Mary."

may take noun phrase complements, sentential complements, or no complement at all. Although all combinations are possible, we demonstrate our analysis with a few instructive examples. Consider the following sentence:

(26) John likes two books about Mary.

In sentence (26), *about* modifies the noun *books* and takes a noun phrase complement. The lexical entry for *about* used in this sentence is given in Figure 5.28 along with updated entries for *books* and *two*. The derived tree pair and derivation tree are given in Figure 5.29. The meaning of *about* makes use of a higher order *and* operator ($\wedge$) defined as $\lambda xPQ.P(x) \wedge Q(x)$.

Because the preposition can take a quantified noun phrase complement, both the lexical entries for prepositions and nouns must include a location for the scope part of a noun phrase to attach.[4] This is accomplished by adding vestigial trees to the lexical entries where the scope of noun phrases may attach either directly or in multiple steps. This solution poses a significant cost in terms of grammar complexity: these lexical entries are no longer tree-local, they are set-local. We solve this problem in Chapter 6 and also address the complex and interesting issue of available scope readings for sentences in which a quantificational noun phrase complement of a preposition nests inside the quantificational noun phrase that the preposition modifies.

The following sentence demonstrates a preposition without a complement modifying a verb phrase:

(27) John sat inside.

The lexical entries for *inside* and *sat* are given in Figure 5.30. The entry for *sat* is isomorphic to the one for *escaped* but is augmented with an additional link to accommodate the preposition. The derived tree pair and derivation tree are given in Figure 5.31.

Finally, the following sentence demonstrates a preposition with a small clause complement modifying a sentence:

(28) With Mary gone, John escaped.

---

[4]Although we could analyze non-quantificational noun phrases without a scope part, it is clear that prepositions can take quantificational noun phrases as complements, making this solution plainly inadequate.

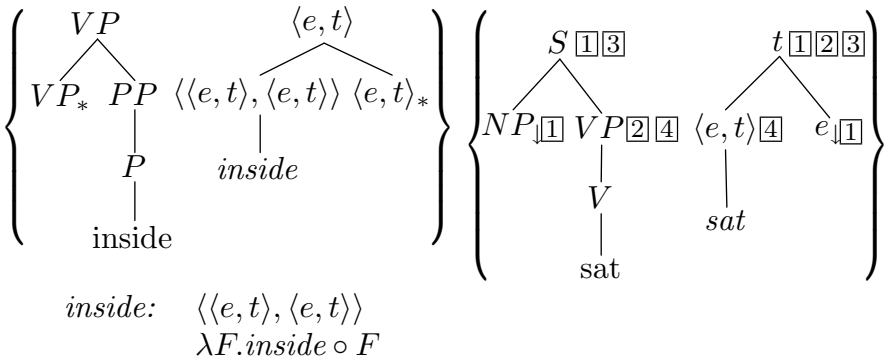


Figure 5.30: Lexical entry for *inside* modeling a verb-phrase modifying pronoun that does not take a complement. Also included is the lexical entry for *sat* which is isomorphic to that for *escaped* and also includes an additional link to accommodate verb phrase modifying prepositions.

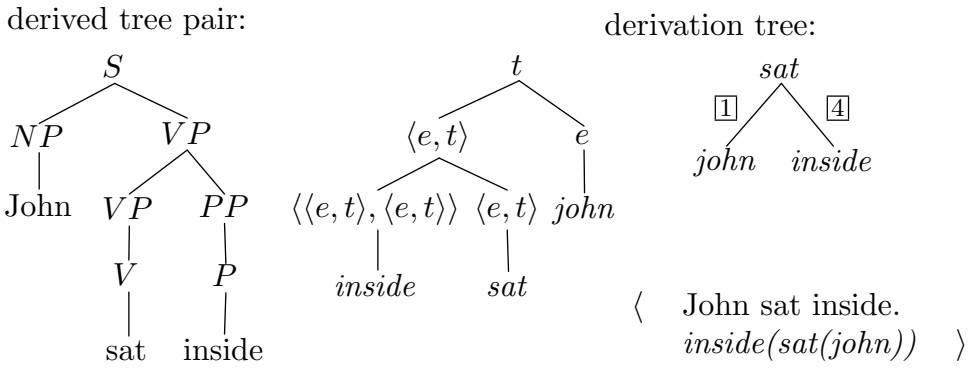


Figure 5.31: Derived tree pair and derivation tree for the sentence "John sat inside."

The lexical entry for *with* is given in Figure 5.32. The derived tree pair and derivation tree are given in Figure 5.33. For simplicity, we do not include an analysis of the small clause.

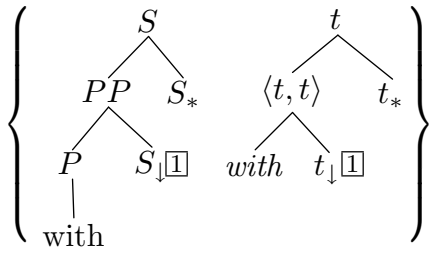


Figure 5.32: Lexical entry for *with* modeling a sentence modifying pronoun that takes a sentential complement.

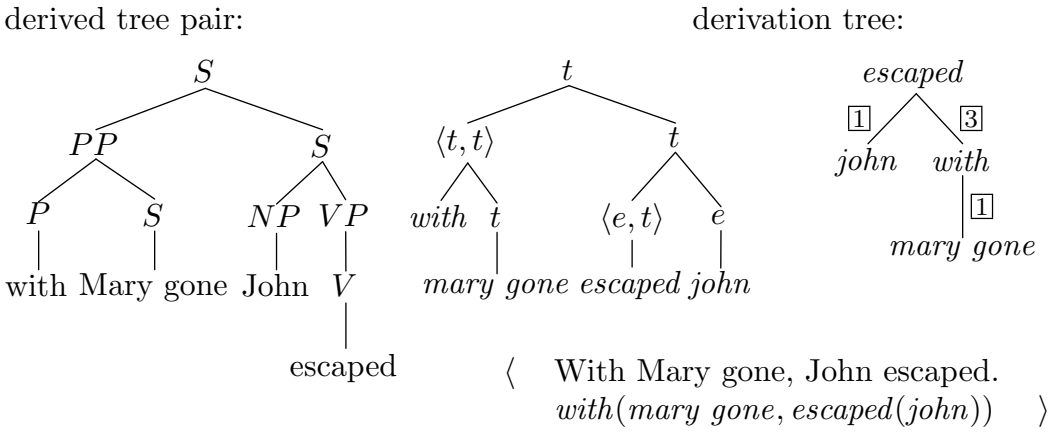


Figure 5.33: Derived tree pair and derivation tree for the sentence "With Mary gone, John escaped."

5.9 Noun Phrase Complements and Relative Clauses

Two common types of subordinate clauses that both modify noun phrases are noun phrase complements and relative clauses. Their analysis shows how subordinate clauses may be modeled and also demonstrates the accurate prediction that movement out of such clauses should not be possible.

Consider the following sentence in which the clause introduced by *that* modifies the noun phrase *the report*, giving additional information about the contents of the

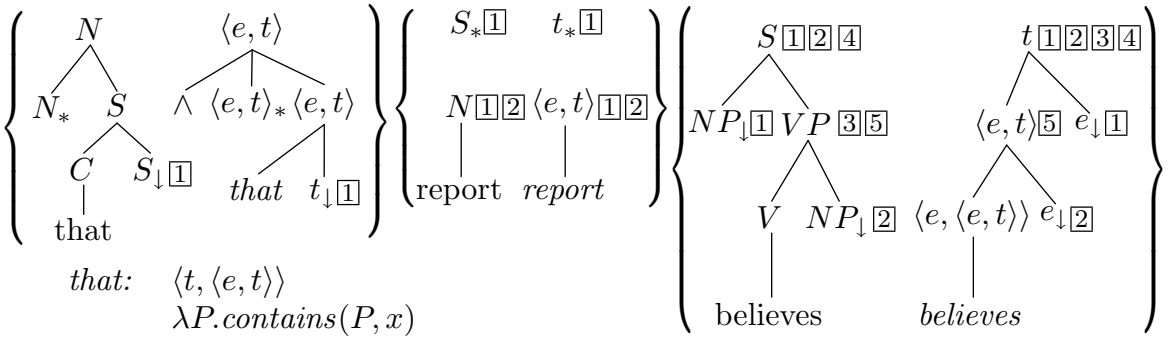


Figure 5.34: Lexical entry for *that* as a noun phrase modifier introducing a subordinate sentence. An updated entry for *report* is also included showing an additional link to which *that* can adjoin. Although it is isomorphic to the entry for *likes*, the lexical entry for *believes* is included for convenience.

noun phrase:

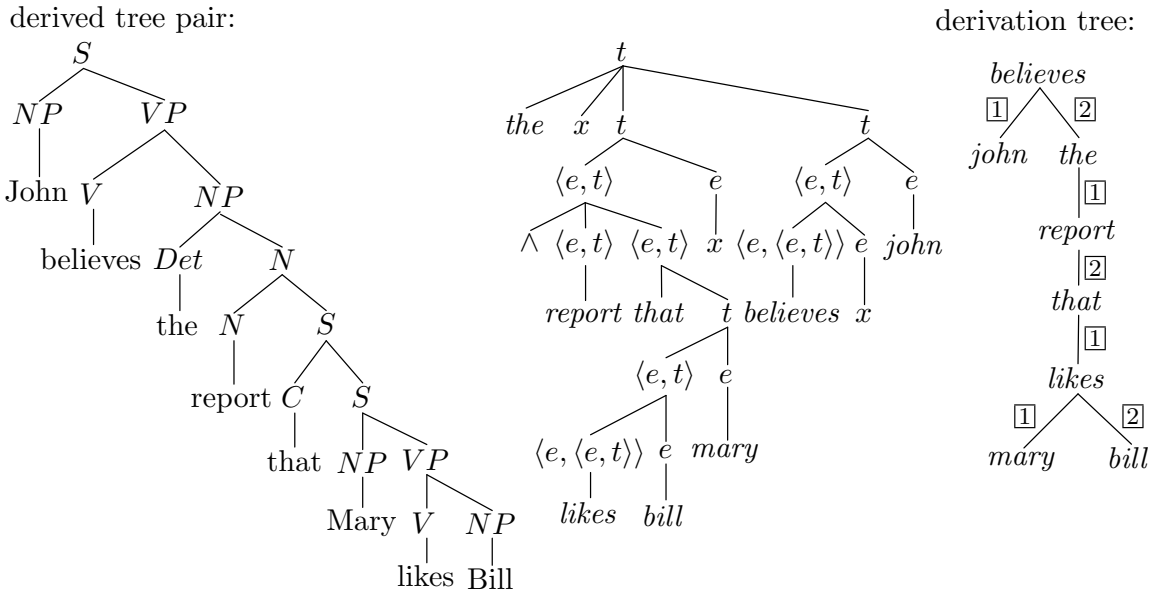
(29) John believes the report that Mary likes Bill.

The lexical entry for *that* along with the other necessary lexical items are provided in Figure 5.34. The derived tree pair and derivation tree for sentence (29) are given in Figure 5.35.

Relative clauses add another level of complexity because the head noun for the subordinate clause is coindexed with a noun in the modifying clause. As a result, the relative pronoun must somehow contribute or connect a variable in both the lower and upper verbs in the sentence. Consider the following sentence:

(30) Bill has two brothers$_i$ who$_i$ t_i like Mary.

This sentence can be analyzed by treating the relative pronoun as a composite of two lexical items, one that is structured similarly to the lexical entry for *that*



⟨ John believes the report that Mary likes Bill.
$the(x, report(x) \wedge contains(likes(bill, mary), x), believes(x, john))$ ⟩

Figure 5.35: Derived tree pair derivation tree for the sentence "John believes the report that Mary likes Bill".

above and introduces the subordinate clause and a second one that abstracts over the coindexed noun phrase. These lexical entries are given in Figure 5.36 along with the lexical entry for *likes*, which is updated to show the REL feature. The lexical entry for *has* is isomorphic to the one for *likes*. Features ensure that a sentence that is the complement of a relative pronoun is in fact expecting to receive one of its arguments from the relative pronoun. The REL feature is represented on the syntax in the lexical entries but would have the same effect if it were included on the corresponding nodes in the semantics or if it were duplicated in both places. The derived tree pair and derivation tree for sentence last are given in Figure 5.37.[5]

[5]Although the type of *who (rel) 1* is $\langle\langle e,t\rangle, \langle\langle e,t\rangle, \langle e,t\rangle\rangle\rangle$, the node label for its substitution nodes is $t_{\downarrow}$. This is because *who (rel) 2* adjoins at t into a sentence and effectively changes the type of

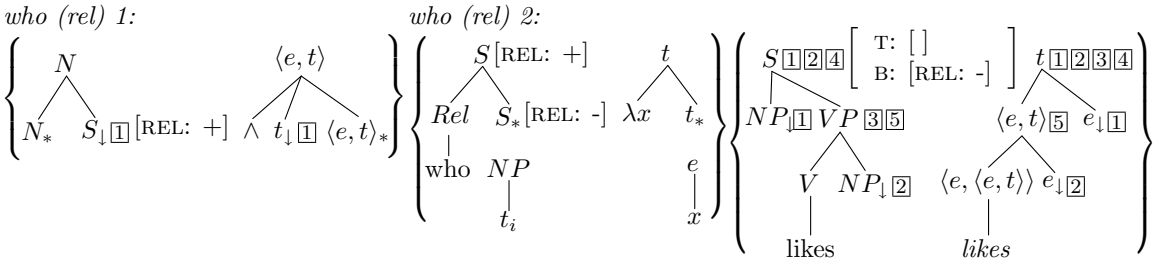
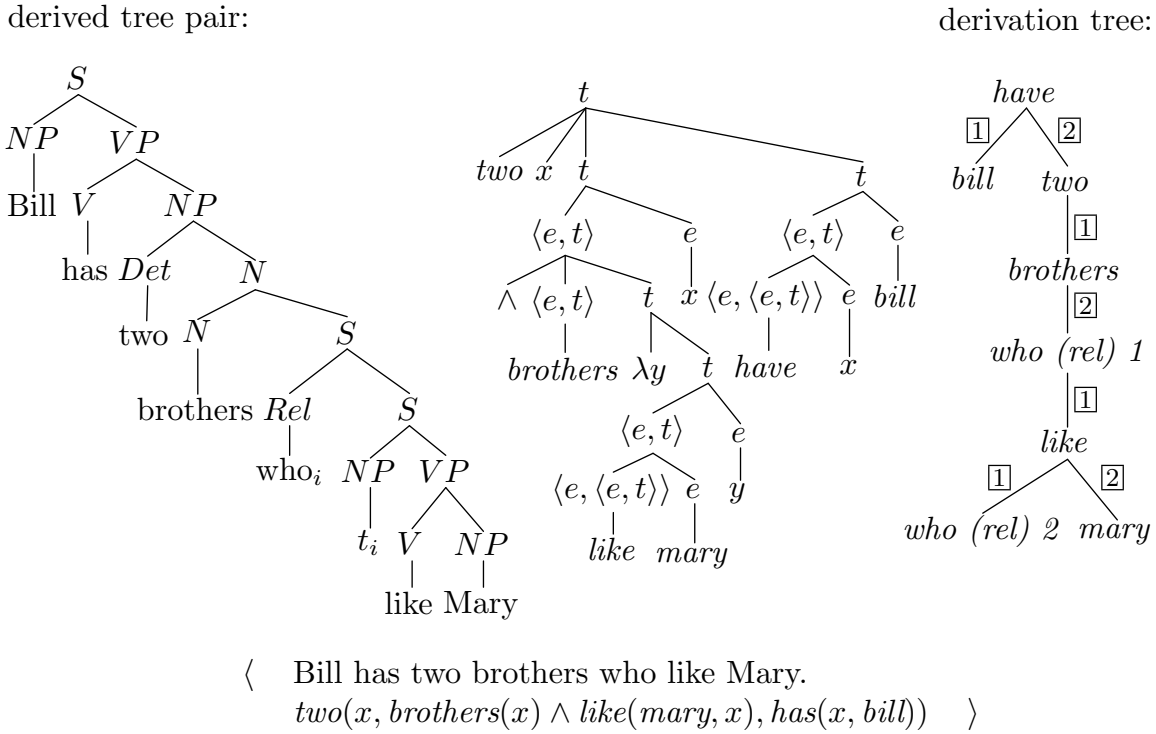


Figure 5.36: Lexical entries that make up the relative pronoun *who* as well as an updated entry for *likes* that demonstrates its REL feature value.



⟨ Bill has two brothers who like Mary.
$two(x, brothers(x) \wedge like(mary, x), has(x, bill))$ ⟩

Figure 5.37: Derived tree pair and derivation tree for the sentence "Bill has two brothers who like Mary."

---

the sentence from t to $\langle e, t\rangle$ by abstracting over one of the arguments. However, the adjunction operation requires that the root and node labels of the tree match the site at which they adjoin. The REL feature can be thought of as a marking that signals that *who (rel) 1* is looking for an argument that exhibits this mismatch between the label on the root node and the actual type of the substituting λ-term.

5.9.1 Restricting Movement Out of Islands

Movement is modeled in this system using multicomponent tree sets in which one tree contains the moved constituent and the other contains the trace that anchors the moved element to its original position. Constraints on movement therefore fall out naturally from the locality constraints imposed by the grammar formalism. Islands—syntactic constructions out of which constituents cannot move—also arise naturally if they do not provide appropriate locations for a moved version of a lexical item to adjoin. That is, the moved portion of the lexical item will need an adjunction site at an S node at which to adjoin. If the lexical item into which the adjunction must take place does not contain such a node, the movement will not be possible. We demonstrate this using the Complex NP constraint. The Complex NP constraint states that constituents cannot move out of a clause modifying a head noun. We use the two constructions introduced above as examples. Consider the following sentences demonstrating unacceptable movement:

(31) *Who$_i$ does John believe the report that Mary likes t_i?

(32) *Who$_j$ does Bill have two brothers$_i$ who$_i$ t_i like t_j?

It is impossible to produce sentence (32) from our lexicon. The problem with the movement arises because the moved noun phrase must adjoin to the verb *likes*. As in our analysis of sentence (29), *likes* substitutes into *that*. This means that the moved element cannot end up higher in the tree than the root of the *likes* tree. It cannot move beyond the clause boundary.[6]

[6]It is notable that our lexicon does not rule out the ungrammatical:

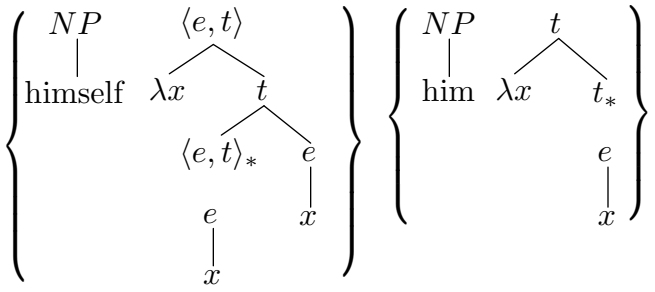


Figure 5.38: Possible lexical entries for *himself* and *him*.

Sentence (32) also cannot be produced for the same reason: the moved noun phrase adjoins to *likes* which in turn substitutes into the relative pronoun. Again, as a result, there is no way for the movement to escape the clause boundary.

5.10 Binding Theory

Binding of pronouns presents a challenging problem for STAG semantics. Unlike the other long-distance dependencies, many binding phenomena resist even the extended domains of locality that STAG can provide. Thus far, long-distance dependencies have been represented using multicomponent tree sets and explicitly named and bound variables within those tree sets. However, pronouns, almost by definition, must be bound by binders that do not originate in their own tree set. A few simple examples help to demonstrate how binding within an elementary tree set quickly breaks down for pronoun binding.

(33) *John believes the report that who$_i$ Mary likes t_i?

A separate mechanism, beyond the scope of this chapter, is needed to govern the availability of indirect question formation.

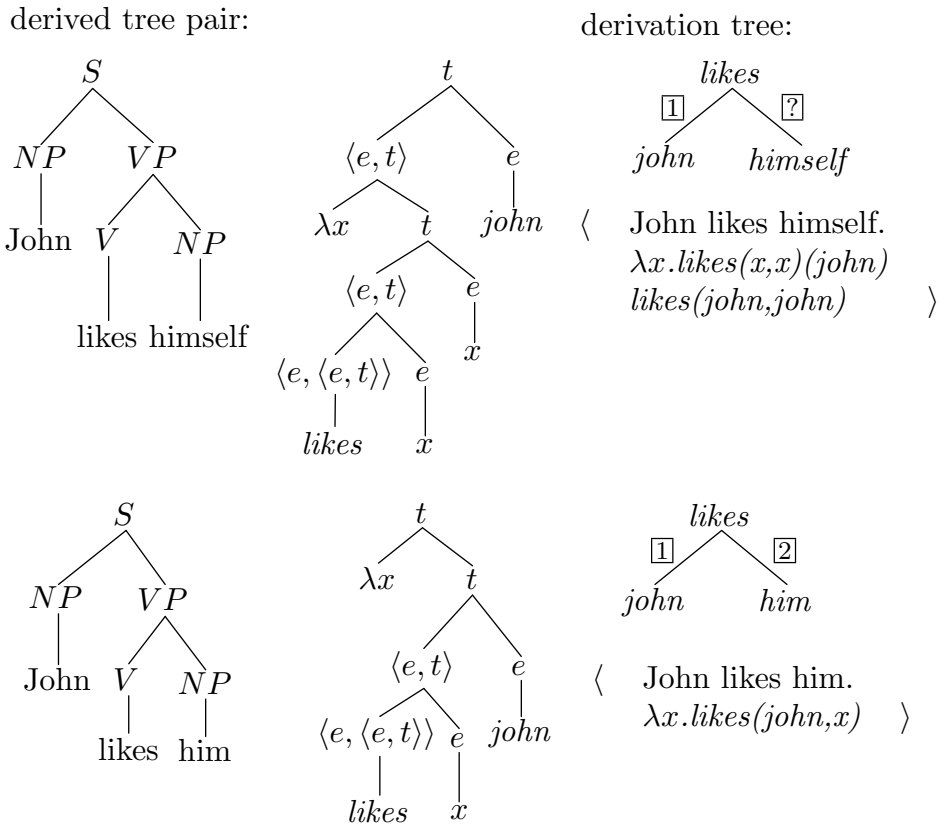


Figure 5.39: Derived tree pairs and derivation trees for the sentences "John likes himself." and "John likes him."

(34) John$_i$ likes himself$_i$.

(35) John$_i$ likes him$_j$. $(i \neq j)$

Sentences (35) and (34) can be analyzed using the lexical entries for *himself* and *him* given in Figure 5.38. The derived tree pairs and derivation trees are given in Figure 5.39. The successful analysis of these two examples relies on the ability of the tree set for the pronoun to capture (or prevent capture) the meaning of *john* and store it in a variable when it adjoins. In the case of *himself*, the lexical entry ensures that the entity in subject position is also interpreted as the object of the verb. In the

case of *him* the lexical entry ensures that some entity from higher in the sentence or from the context occurs in object position.

Even for these simple sentences, the presented analyses have several drawbacks. First, the tree set for *himself* must adjoin at an $\langle e, t\rangle$ node in the semantics, which diverges from the usual structure of noun phrases in our grammar and for which we would have to add a special link to each tree to which it could adjoin. Second, the lexical entry for *him* constrains the location at which the λ abstraction can occur according to our usual movement rules. Further empirical examples would have to be tested to determine whether this prediction is desirable. However, even though we may discard these first objections, more complex sentences demonstrate that this method of capturing the desired entity and representing it with a variable breaks down. Consider the following example:

(36) John$_i$ likes the book about himself$_i$.

In sentence (36) the reflexive must adjoin to the tree set for the preposition in any reasonable syntactic and semantic analysis. However, in order to "capture" *john*, the tree containing the λ abstraction must adjoin to the main verb *likes*. Without dropping all locality constraints, this forces a highly improbable and costly analysis.[7] We therefore pursue a different strategy.

[7]Assuming that we added a vestigial tree $\langle e, t\rangle_*$ to the tree set for *about*, we could keep the λ abstraction of *himself* free until later in the derivation. However, *about* must then adjoin to *two*, which would also need an additional vestigial tree $\langle e, t\rangle_*$. With that tree added, then by modifying the links in the *likes* tree we could allow the λ abstraction to adjoin at the $\langle e, t\rangle$ node of *likes*. This proliferation of vestigial trees to thwart the locality requirements is clearly less than desirable.

To handle this case and others, we use a different method for representing the semantics of bound pronouns. Rather than using explicitly named variables and corresponding lambda abstractions, we use an alternative formulation of the lambda calculus called De Bruijn notation that dispenses with explicit naming. The following section provides an introduction to De Bruijn notation before returning to its application to binding theory.

5.10.1 De Bruijn Notation

De Bruijn notation is an alternative notation for the lambda calculus that dispenses with free variables by replacing explicitly named variables with natural numbers that indicate the distance (in terms of number of enclosing λ-terms) of the binding λ. Although the de Bruijn notation is equivalent to the usual lambda calculus notation, the absence of free variables makes it possible to generate lexical items that are well-formed but that include a variable that will be bound by a λ that occurs in an entirely separate lexical item. This idea, in which the variable "cares" not about the specific λ that binds it but rather about some notion of the distance of the λ that binds it meshes extremely well with the way in which pronomials find their referents.

The notation can be understood informally by comparison to standard notation:

$$\lambda.\lambda.2 \text{ is equivalent to } \lambda xy.x$$

This equivalence holds because the outermost λ is the second λ enclosing the variable 2. If the variable were 1 instead, then the term would be equivalent to $\lambda xy.y$. When a β-reduction is performed, the substituted variable replaces all variables bound by the

given λ, however the value of the substituted variable may itself have to be augmented in order to remain bound by its binder. In addition, other variables in the term may have to be decremented because of the reduction of one of the enclosing λs.

Formally, lambda terms in de Bruijn notation are defined [Hankin, 2004] inductively as the least set in which:

(i) any natural number greater than zero is a term,

(ii) if M and N are terms, then (MN) is a term,

(iii) if M is a term, (λM) is a term.

The β rule is replaced by:

$$(\lambda P)Q = P[1 := Q]$$

where:

$$\begin{aligned}
n[m := N] &\equiv \begin{cases} n & \text{if } n < m \\ n-1 & \text{if } n > m \\ rename_{n,1}(N) & \text{if } n = m \end{cases} \\
(M_1 M_2)[m := N] &\equiv (M_1[m := N])(M_2[m := N]) \\
(\lambda M)[m := N] &\equiv \lambda(M[m+1 := N]) \\
rename_{m,i}(j) &\equiv \begin{cases} j & \text{if } j < i \\ j+m-1 & \text{if } j \geq i \end{cases} \\
rename_{m,i}(N_1 N_2) &\equiv rename_{m,i}(N_1) rename_{m,i}(N_2) \\
rename_{m,i}(\lambda N) &\equiv \lambda(rename_{m,i+1}(N))
\end{aligned}$$

Some explication helps to elucidate these rules. The first rule governs the updating of variables themselves. In the first rule, m is a counter of how many λs deep the substitution is nested. If the counter equals the value of the variable, then we substitute for it using the *rename* rules. If the counter is greater than the variable, then the relative relationship of the variable to its binder is not affected by the reduction so it remains unchanged. If the counter is less than the variable, then one of the λs between the variable and its binder is being reduced, so the variable must be decremented to remain bound by the same λ. The second rule simply passes the substitution to both terms in an application. The third rule increments the counter as a λ is entered by the substitution.

The first *rename* rule handles the case where a variable is substituted. The variable itself must be renamed to preserve its binding. Here, both m and i are counters. The level of nesting of the variable to be substituted within the term to be substituted is counted by i. If the variable is bound within the term ($j < i$), then it is not changed. However, if it is bound outside of the term being β-reduced, it is incremented by the new levels of nesting being counted by m. The remaining *rename* rules simply pass the renaming through the term that is being substituted, augmenting the counter i for that term as necessary.

Although we include the β rule for completeness, rather than applying the de Bruijn β rule, for readability we will convert all of our derived semantics to standard notation and use standard β-reduction.

We benignly extend the de Bruijn notation in several respects. First, in certain cases we specify variables using inequalities that symbolize ranges of natural numbers

rather than single natural numbers. In these cases we intend this notation to indicate that the variable may take on any single value within the range. Its value is not to be interpreted as a range of numbers. The convention in de Bruijn notation for numbers larger than the number of nesting λs is that they index into an ordered list of free variables. Thus we allow readings where variables remain free. We make use of this in our analyses to model pronomials finding referents outside of the sentence. We leave open the possibility of treating the referents in the context in a way that makes a greater range of values meaningful by picking out different referents in the discourse context.

Second, in some cases lexical items in our grammar contain lambda terms that use the standard variable notation. These cases are those in which both the binder and the bound variable appear within the same lexical item. We assume that these λs bind only the named variables for which they are defined and are ignored as potential binders by the variables in de Bruijn notation. After the derivation is complete and before the lambda terms are reduced, a conversion is performed so that all lambda terms share the same notation. It is most common to use the standard notation for readability and to convert to the de Bruijn notation to simplify processing. In our case, we wish to maintain readability so we convert the de Bruijn λs and numbers to standard notation.

The conversion from de Bruijn to standard notation is defined as follows.

$$
\begin{array}{lcll}
S\ i\ (x_1, \ldots, x_n) & = & x_i & \\
S(\lambda M)\ (x_1, \ldots, x_n) & = & \lambda y.(S\ M\ (y, x_1, \ldots, x_n)) & \mathsf{FreshVar}(y) \\
S(MN)\ (x_1, \ldots, x_n) & = & (S\ M\ (x_1, \ldots, x_n))(S\ N\ (x_1, \ldots, x_n)) &
\end{array}
$$

An example helps to elucidate the application of the conversion:

$S((\lambda\lambda\lambda.321)(\lambda\lambda.2))\ ()$

$(S\ \lambda\lambda\lambda.321\ ())(S\ \lambda\lambda.2\ ())$

$(\lambda y_1.S\ \lambda\lambda.321\ (y_1))(S\ \lambda\lambda.2\ ())$

$(\lambda y_1 \lambda y_2.S\ \lambda.321\ (y_2, y_1))(S\ \lambda\lambda.2\ ())$

$(\lambda y_1 \lambda y_2 \lambda y_3.(S\ 32\ (y_3, y_2, y_1)(S\ 1\ (y_3, y_2, y_1)))(S\ \lambda\lambda.2\ ())$

$(\lambda y_1 \lambda y_2 \lambda y_3.(S\ 3\ (y_3, y_2, y_1))(S\ 2\ (y_3, y_2, y_1))(S\ 1\ (y_3, y_2, y_1)))(S\ \lambda\lambda.2\ ())$

$(\lambda y_1 \lambda y_2 \lambda y_3.y_1(S\ 2\ (y_3, y_2, y_1))(S\ 1\ (y_3, y_2, y_1)))(S\ \lambda\lambda.2\ ())$

$(\lambda y_1 \lambda y_2 \lambda y_3.y_1 y_2(S\ 1\ (y_3, y_2, y_1)))(S\ \lambda\lambda.2\ ())$

$(\lambda y_1 \lambda y_2 \lambda y_3.y_1 y_2 y_3)(S\ \lambda\lambda.2\ ())$

$(\lambda y_1 \lambda y_2 \lambda y_3.y_1 y_2 y_3)(\lambda y_4.S\ \lambda.2\ (y_4))$

$(\lambda y_1 \lambda y_2 \lambda y_3.y_1 y_2 y_3)(\lambda y_4 \lambda y_5.S\ 2\ (y_5, y_4))$

$(\lambda y_1 \lambda y_2 \lambda y_3.y_1 y_2 y_3)(\lambda y_4 \lambda y_5.y_4)$

With the conversion complete, we can do standard β-reduction to produce the final result:

$(\lambda y_1 \lambda y_2 \lambda y_3.y_1 y_2 y_3)(\lambda y_4 \lambda y_5.y_4)$

$\lambda y_2 \lambda y_3.(\lambda y_4 \lambda y_5.y_4) y_2 y_3$

$\lambda y_2 \lambda y_3.(\lambda y_5.y_2) y_3$

$\lambda y_2 \lambda y_3.y_2$

The conversion from mixed notation to standard notation is similar. The additional rules simply pass over the parts of the term that are already in standard notation:

$$
\begin{array}{lcll}
S\ i\ (x_1, \ldots, x_n) & = & x_i & \\
S(\lambda M)\ (x_1, \ldots, x_n) & = & \lambda y.(S\ M\ (y, x_1, \ldots, x_n)) & \mathsf{FreshVar}(y) \\
S(MN)\ (x_1, \ldots, x_n) & = & (S\ M\ (x_1, \ldots, x_n))(S\ N\ (x_1, \ldots, x_n)) & \\
S\ x\ (x_1, \ldots, x_n) & = & x & \\
S(\lambda x M)\ (x_1, \ldots, x_n) & = & \lambda x.(S\ M\ (x_1, \ldots, x_n)) &
\end{array}
$$

Again, an example helps to elucidate its application:

$S(\lambda x_1 \lambda\lambda\lambda z.1x2z)\ ()$

$\lambda x_1.(S(\lambda\lambda\lambda z.1x2z)\ ())$

$\lambda x_1 \lambda y_1.(S(\lambda\lambda z.1x2z)\ (y_1))$

$\lambda x_1 \lambda y_1 \lambda y_2.(S(\lambda z.1x2z)\ (y_2, y_1))$

$\lambda x_1 \lambda y_1 \lambda y_2 \lambda z.(S(1x2\ (y_2, y_1))S(z\ (y_2, y_1)))$

$\lambda x_1 \lambda y_1 \lambda y_2 \lambda z.(S(1x\ (y_2, y_1))S(2\ (y_2, y_1))S(z\ (y_2, y_1)))$

$\lambda x_1 \lambda y_1 \lambda y_2 \lambda z.(S(1\ (y_2, y_1))S(x\ (y_2, y_1))S(2\ (y_2, y_1))S(z\ (y_2, y_1)))$

$\lambda x_1 \lambda y_1 \lambda y_2 \lambda z.y_2 S(x\ (y_2, y_1))S(2\ (y_2, y_1))S(z\ (y_2, y_1)))$

$\lambda x_1 \lambda y_1 \lambda y_2 \lambda z.y_2 x S(2\ (y_2, y_1))S(z\ (y_2, y_1)))$

$\lambda x_1 \lambda y_1 \lambda y_2 \lambda z.y_2 x y_1 S(z\ (y_2, y_1)))$

$\lambda x_1 \lambda y_1 \lambda y_2 \lambda z.y_2 x y_1 z$

5.10.2 Using De Bruijn Notation to Model Binding Theory

With de Bruijn notation available to us we can update our analysis of pronouns. As with other variables in our grammar, we use de Bruijn indices to represent entities. In the case of a pronoun, a de Bruijn index represents the pronoun. Unlike in our earlier

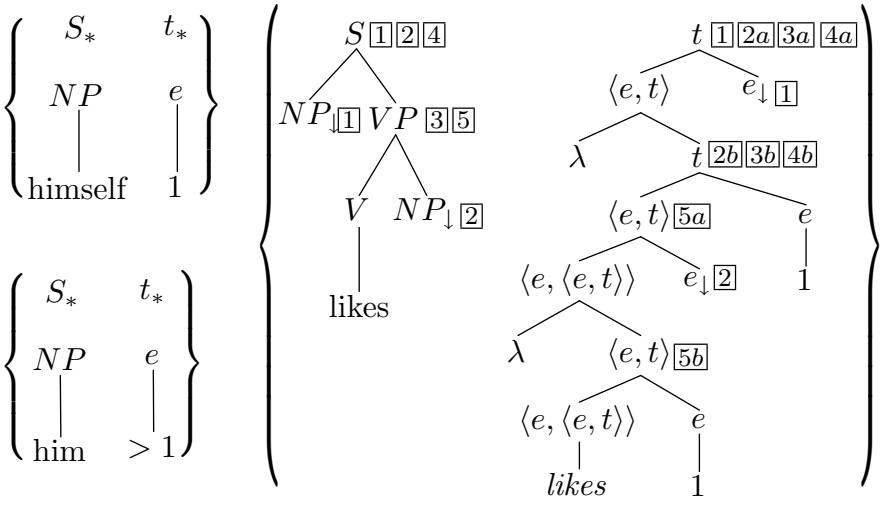


Figure 5.40: The lexical entries for *himself* and *him* using de Bruijn indices. The lexical entries for verbs and other lexical items that take noun phrase/entity arguments are also updated to include λs that can bind any de Bruijn indices that may be contained in their entity arguments. Note that in the semantics of *likes* certain links, such as 2 are annotated in some locations with additional subscripts (2*a* or 2*b*). These annotated link locations are to be interpreted as alternative link locations for the primary link; one or the other of the subscripted locations must be used in an adjunction, but not both.

analysis, we do not require the λ that will eventually bind the variable to originate in the same lexical item. Instead, we augment the lexical entries that expect to receive entities by substitution with λs designed to make use of the de Bruijn indices that may appear in those entities. Figure 5.40 contains the new lexical entries for *himself* and *him* as well as the updated entry for *likes* that makes use of the de Bruijn indices in the pronouns.

The lexical entry for *likes* bears some explanation. At each location at which an entity argument is expected, the tree is expanded with a λ abstraction that can potentially bind de Bruijn indices that occur within that argument entity. When the expansion entails splitting a node at which a link was present, that link location is

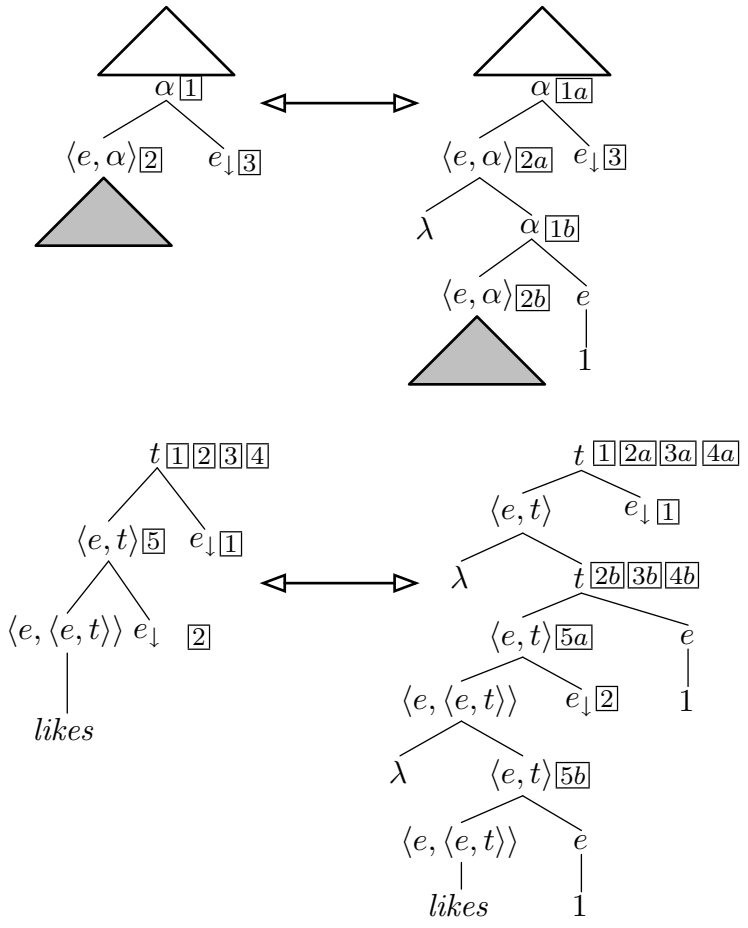


Figure 5.41: A schematic representation of the transformation of a simplified tree into its full form as an expanded tree and an example of the application of the rule to the elementary tree for *likes*. Note that the transformation is bidirectional.

split into two alternative link locations (subscripted with lowercase letters). Those subscripted link locations are to be interpreted as alternative locations within their link. One and only one of the alternative locations of a link must be used when an adjunction takes place at that link. For instance, if a tree set adjoins at link 2, it must adjoin at all of locations marked 2 and also at either 2a or 2b.[8]

[8]In the tree for *likes*, link 1 does not include a link site on the lower t node created by the expansion. This is because of an assumed dominance relationship between the trees in the tree set that adjoin at link 1. We explore making these dominance relationships explicit in Chapter 6.

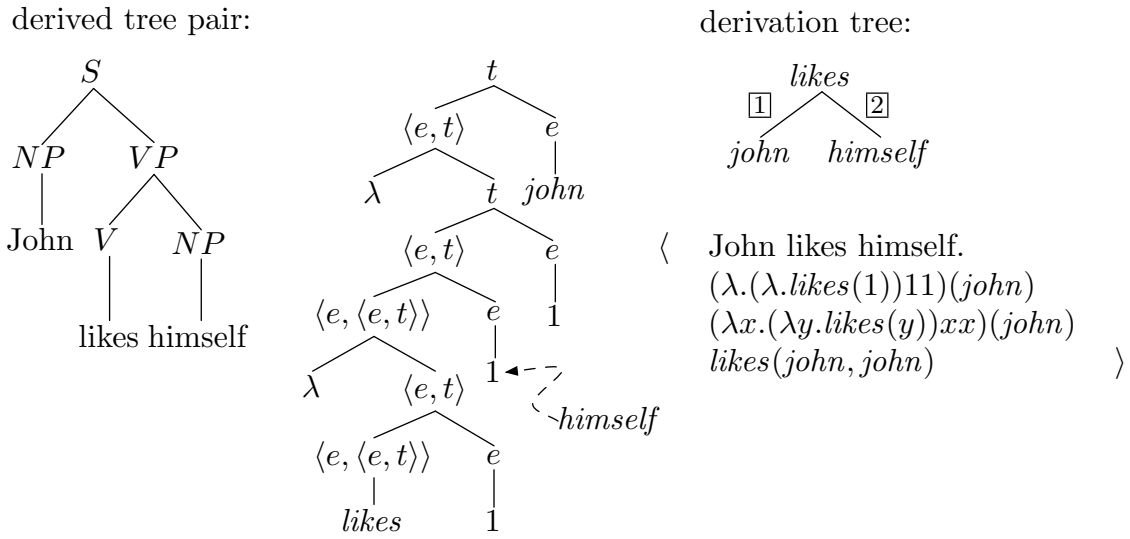


Figure 5.42: The derived tree pair and derivation tree for the sentence "John likes himself." Both the conversion from mixed notation to standard notation and the β-reduction of the derived semantics are shown. Where link 2 is indicated in the derived tree either link 2*a* or link 2*b* may be used without altering the derived semantics produced.

The expansion of semantic trees expecting entity arguments is schematic and can be applied mechanically. Figure 5.41 shows the operation of the expansion schematically. In this section we show the trees in expanded form. However, elsewhere we use the unexpanded versions as a shorthand with the understanding that all trees that fit the schema do in fact include the expanded structure.

We are now prepared to analyze the simple sentences addressed in the introduction to this chapter, repeated here for convenience:

(37) John$_i$ likes himself$_i$.

(38) John$_i$ likes him$_j$. $(i \neq j)$

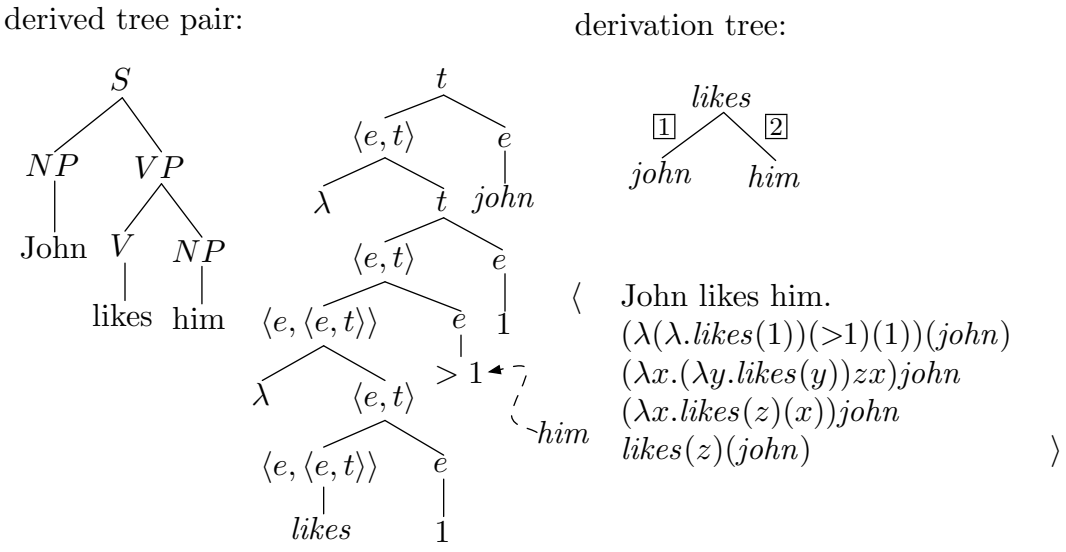


Figure 5.43: The derived tree pair and derivation tree for the sentence "John likes him." Both the conversion from mixed notation to standard notation and the β-reduction of the derived semantics are shown. Where link 2 is indicated in the derived tree either link 2*a* or link 2*b* may be used without altering the derived semantics produced.

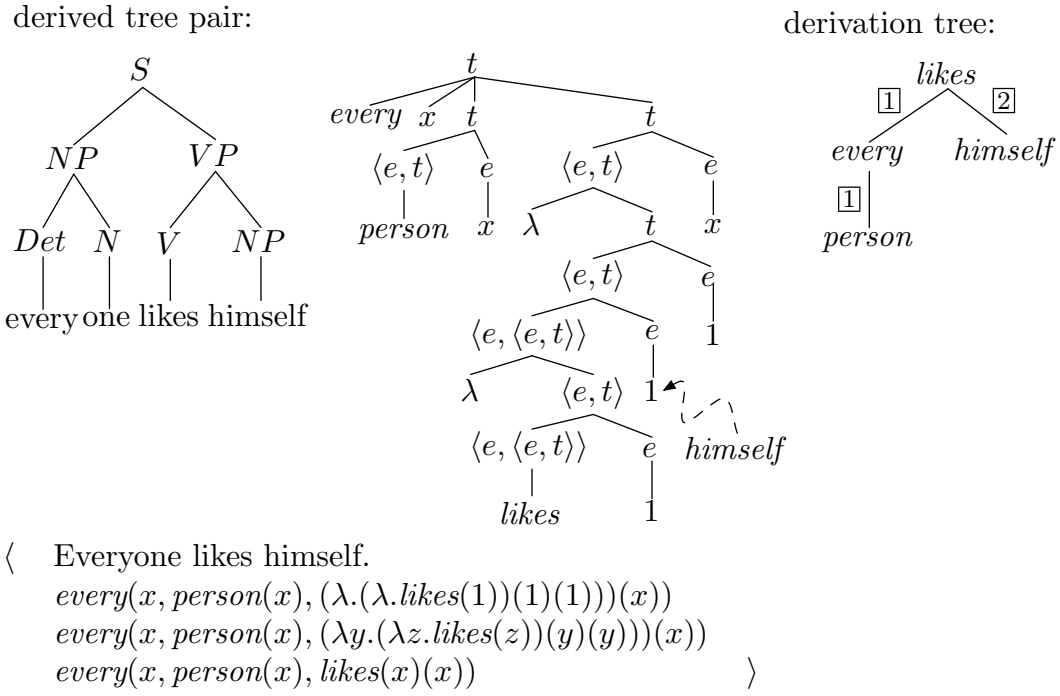
The derived tree pair and derivation tree for sentence (37) are given in Figure 5.42. Figure 5.43 gives the derived tree pair and derivation tree for sentence (38).

The interaction of the de Bruijn indices for pronouns with the analyses for quantifiers are unproblematic in simple sentences. The derived tree pair and derivation for sentence (39) are given in Figure 5.44:

(39) Everyone likes himself.

The addition of de Bruijn notation also permits analysis of more complex sentences containing pronouns. Consider the following sentence:

(40) Mary told John$_i$ about himself$_i$.



⟨ Everyone likes himself.
every$(x, person(x), (\lambda.(\lambda.likes(1))(1)(1)))(x))$
every$(x, person(x), (\lambda y.(\lambda z.likes(z))(y)(y)))(x))$
every$(x, person(x), likes(x)(x))$ ⟩

Figure 5.44: The derived tree pair and derivation tree for the sentence "Everyone likes himself."

The lexical entries for *about* and *told* are given in Figure 5.45. Once again we are presented with the complication of needing non-tree-local adjunction sites for the scope parts of noun phrases to prepositions that take noun phrase complements. As in Section 5.8, we address this here with the addition of vestigial trees in the lexical entries for the prepositions to receive these scope parts but again note that we will resolve this problem in a more satisfactory (and with less consequence to the complexity of our formalism) in Chapter 6. A related issue is that link $\boxed{5}$ in the lexical entry for *told* now also requires additional locations. These locations are marked as $\boxed{5a'}$ and $\boxed{5b'}$. To use link $\boxed{5}$ a tree set must use either location $\boxed{5a}$ or $\boxed{5b}$ and either

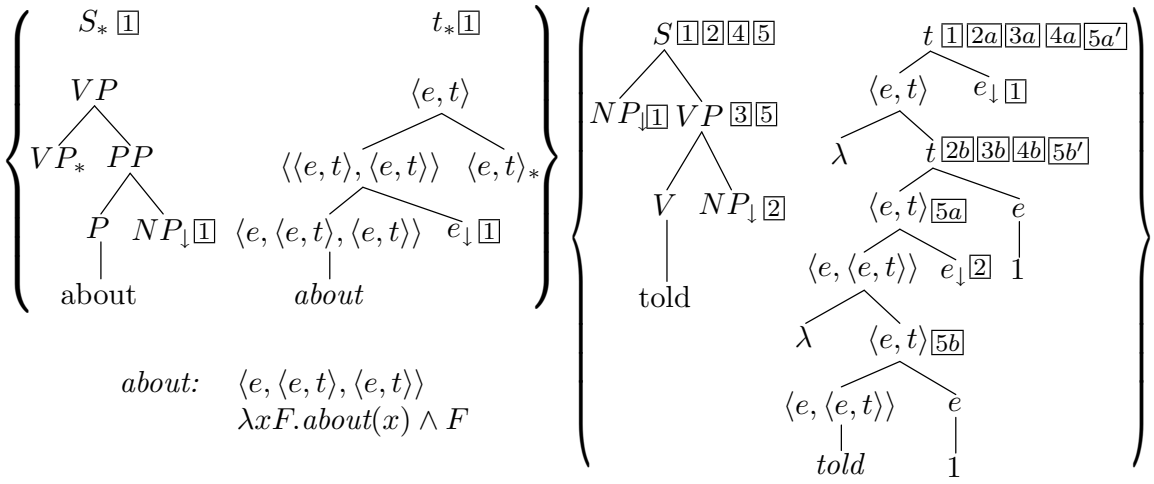


Figure 5.45: Lexical entries for *about* as a verb phrase modifier and *told*. The entry for *told* is isomorphic to the one for *likes*. An additional link location has been added to link 5 to account for the preposition taking a noun phrase argument (in contrast to *inside*, the only verb phrase modifying pronoun we have introduced thus far). Because this link location is on an expanded node, it generates two locations in the expanded tree. These locations are marked with links 5a′ and 5b′.

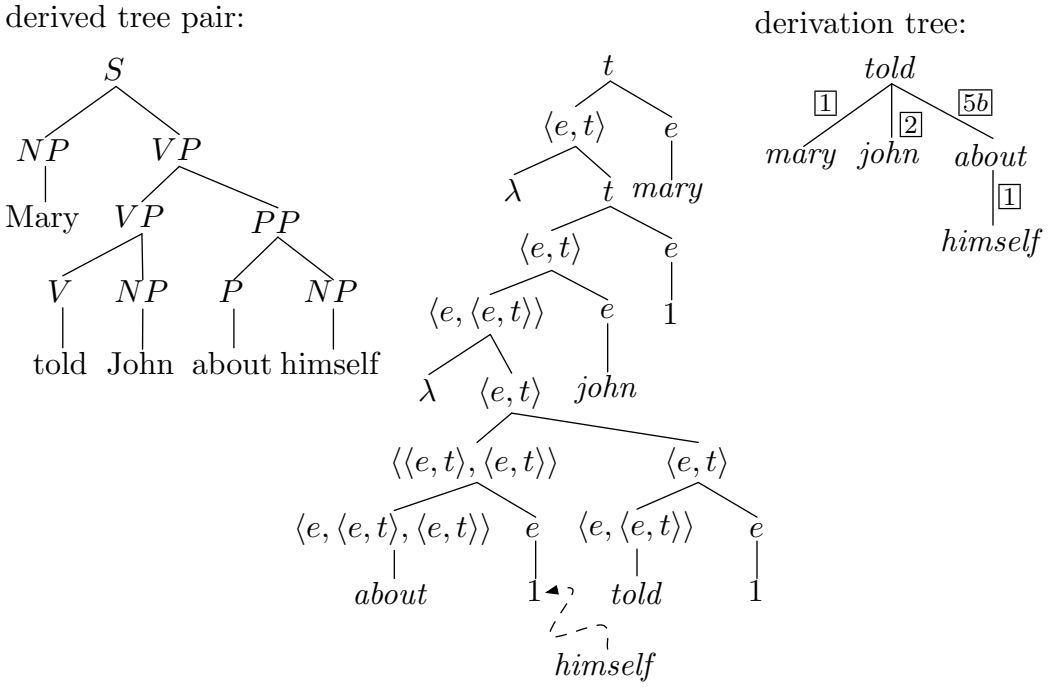
location 5a′ or 5b′.[9]

The derived tree pair and derivation tree for sentence (40) are given in Figure 5.46. Because the preposition adjoins at link 5b, the lower of the two locations for link 5, the de Bruijn index contributed by *himself* corresponds to the λ that takes the object entity (*john*) as its argument.[10] This produces the desired reading.

We can contrast this with the equally acceptable sentence in which the reflexive pronoun refers to the subject entity rather than the object entity:

[9]The careful reader may observe that the addition of these locations to link 5 undermines the earlier use of the analogous link by *inside* in the sentence "John sat inside." Either a separate link with only two locations will be required or the addition of vestigial trees to verb phrase modifying prepositions will have to be made across the board regardless of whether they accept noun phrase arguments. Because we anticipate resolving the issue by a different method in Chapter 6, we do not take a position on this issue here.

[10]Because *himself* is non-quantificational it doesn't matter whether its vestigial t_* tree adjoins at link 5a′ or 5b′, so we do not specify this in the derivation tree.

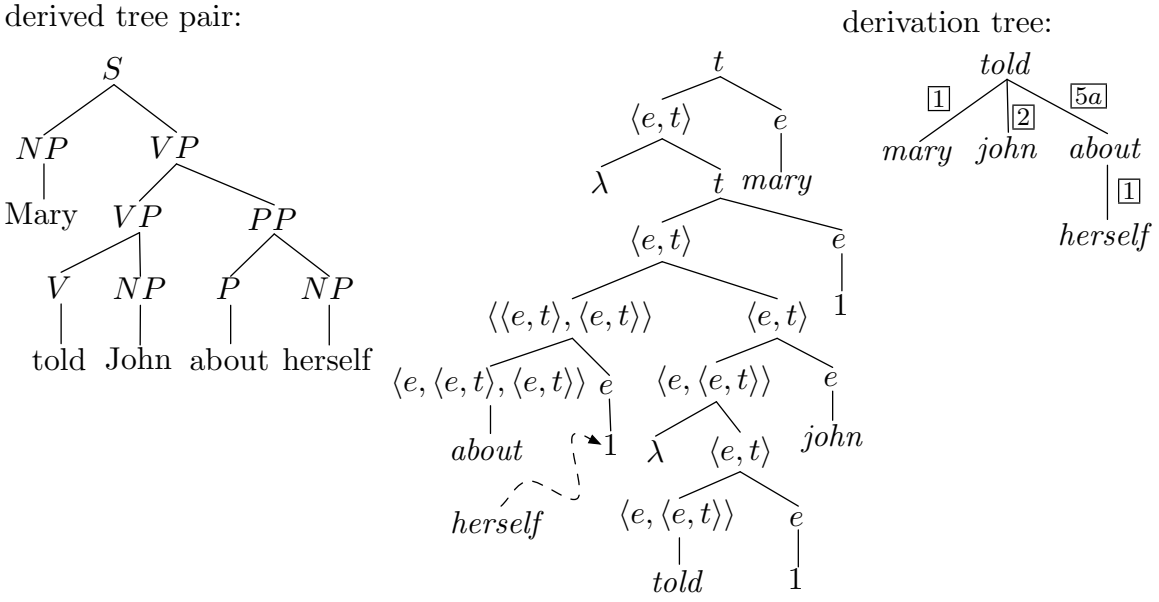


⟨ Mary told John about himself.
$(\lambda.(\lambda.(about(1))(told(1)))(john)(1))(mary)$
$(\lambda.(\lambda.((\lambda xFy.about(x) \wedge F(y))(1))(told(1)))(john)(1))(mary)$
$(\lambda v.(\lambda w.((\lambda xFy.about(x) \wedge F(y))(w))(told(w)))(john)(v))(mary)$
$(\lambda v.(\lambda w.\lambda y.about(w) \wedge told(w)(y))(john)(v))(mary)$
$(\lambda v.about(john) \wedge told(john)(v))(mary)$
$about(john) \wedge told(john)(mary)$ ⟩

Figure 5.46: The derived tree pair and derivation tree for the sentence "Mary told John about himself."

(41) Mary$_i$ told John about herself$_i$.

The lexical entry for *herself* is the same as that for *himself* except for an assumed difference in a gender feature, which we do not show. The derived tree pair and derivation tree for sentence (41) are given in Figure 5.47. In this case, the preposition adjoins at $\boxed{5a}$ so that the de Bruijn index introduced by the pronoun corresponds



⟨ Mary told John about herself.
$(\lambda.(((about(1))((\lambda.told(1))(john)))(1))(mary)$
$(\lambda v.(((about(v))((\lambda w.told(w))(john)))(v))(mary)$
$(\lambda v.((about(v))(told(john)))(v))(mary)$
$((about(mary))(told(john)))(mary)$
$about(mary) \wedge told(john)(mary)$ ⟩

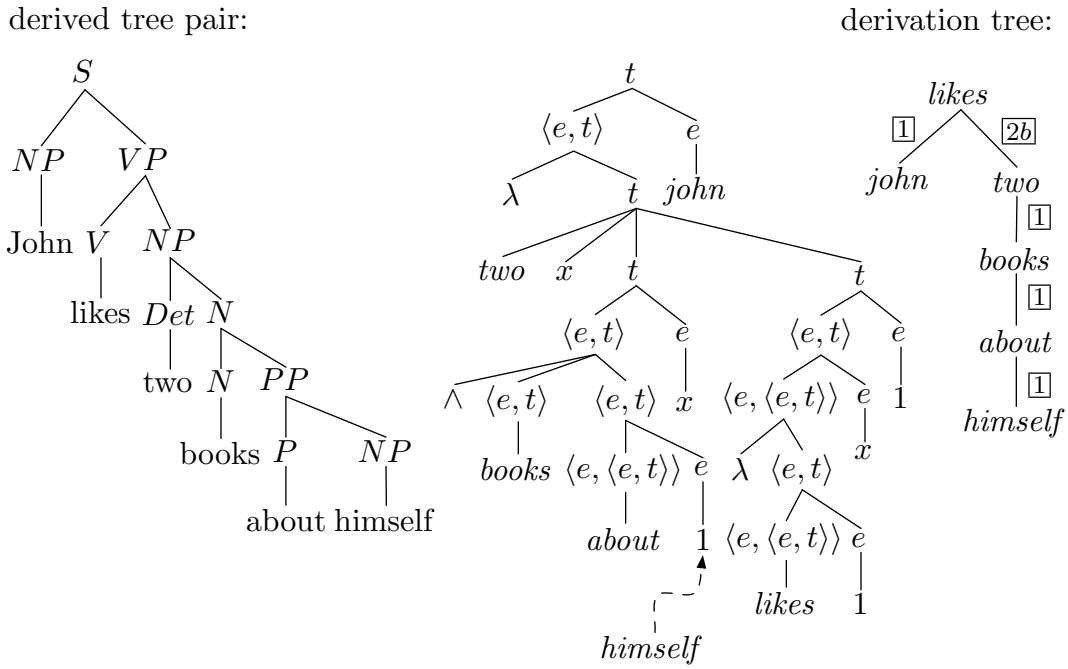
Figure 5.47: The derived tree pair and derivation tree for the sentence "Mary told John about herself."

to the λ that takes the subject entity as an argument. Again, the desired reading is produced.[11] Note that the possibility of two attachment points also predicts the availability of two readings for sentence (42).

(42) John$_i$ told Bill$_i$ about himself$_{i/j}$.

The last several examples dealt with cases in which the pronoun appeared in a VP-modifying prepositional phrase. Similar methods also allow us to account for

[11]We assume that a gender mismatch in the features on *john* (resp. *mary*) and *herself* (resp. *himself*) will prevent the reading in which they coindex.



⟨ John likes two books about himself.
$\lambda.two(x, book(x) \wedge about(1)(x), (\lambda.likes(1))(x)(1)))(john)$
$\lambda y.two(x, book(x) \wedge about(y)(x), (\lambda z.likes(z))(x)(y)))(john)$
$two(x, book(x) \wedge about(john)(x), likes(x)(john)$ ⟩

Figure 5.48: The derived tree pair and derivation tree for the sentence "John likes two books about himself."

NP-modifying prepositional phrases that contain pronouns. The derived tree pair and derivation tree for sentence (44) are given in Figure 5.48.[12]

(44) John$_i$ likes two books about himself$_i$

[12]Note that correct binding for *himself* requires that the quantifier in object position to adjoin at the lower t node ($\boxed{2b}$) in the main verb. If the object quantifier is allowed to adjoin at the higher t node ($\boxed{2a}$) we would predict the acceptability of sentences such as (43), which are generally thought to be unacceptable.

(43) ??John$_i$ thinks Mary likes two books about himself$_i$.

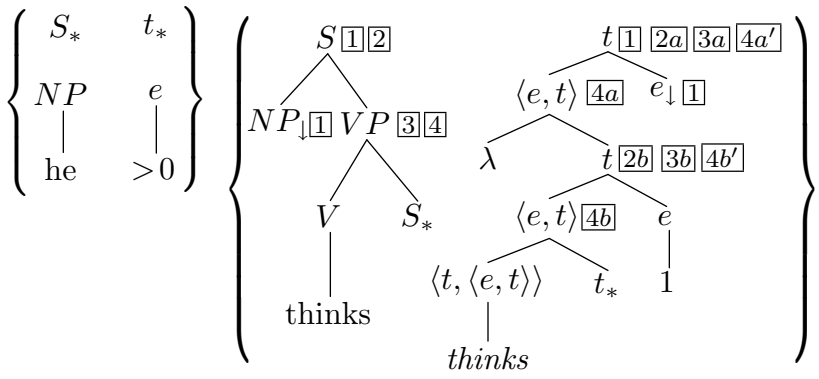


Figure 5.49: Lexical entries for *he* and *thinks*.

The current lexicon correctly predicts the grammaticality and ungrammaticality of the following three sentences as well.

(45) *John$_i$ thinks Mary likes himself$_i$.

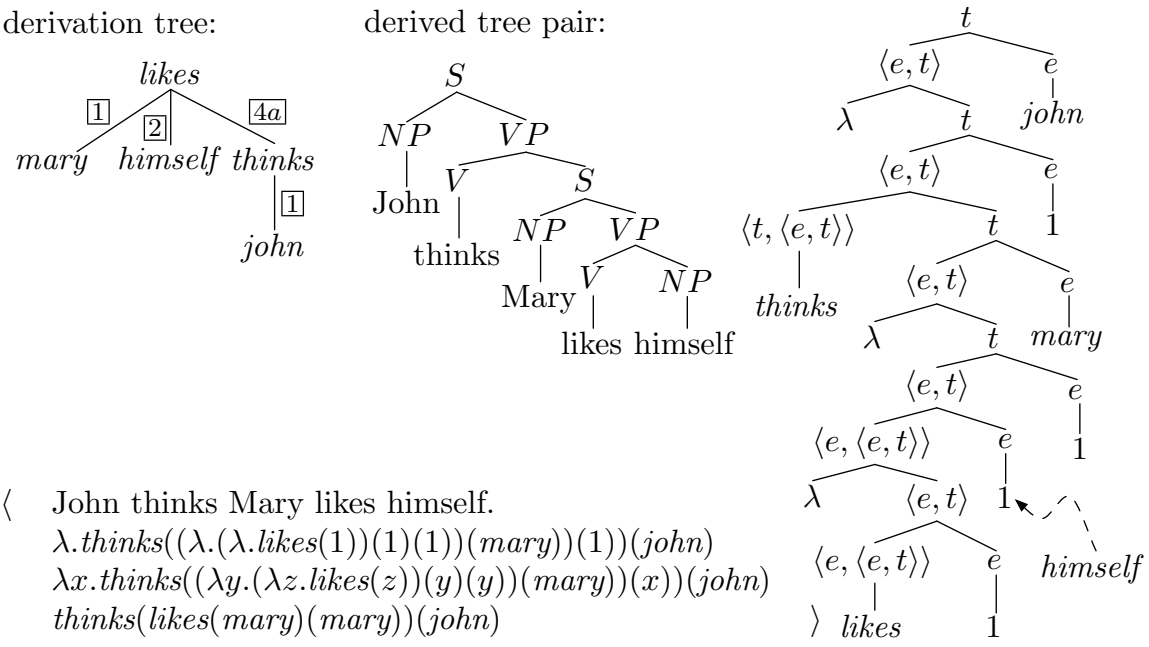
(46) John$_i$ thinks Mary$_k$ likes him$_{i/j}$ ($i \neq j \neq k$).

(47) John$_i$ thinks he$_j$ likes himself$_j$.

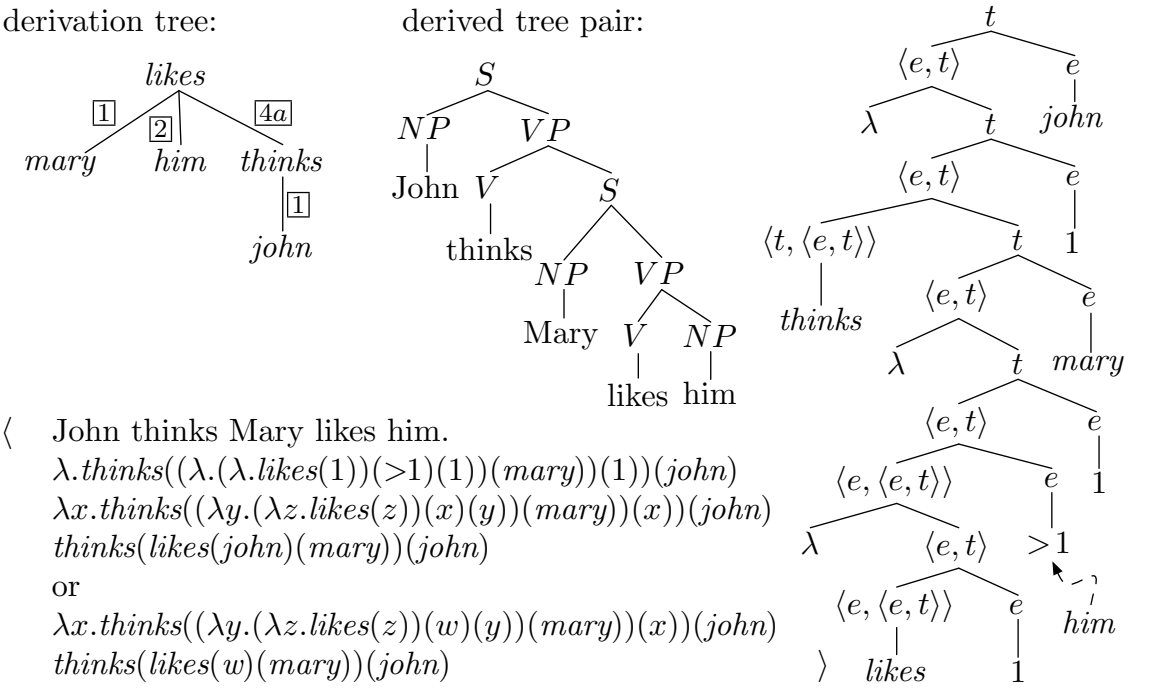
The additional lexical entries for *thinks* and *he* are given in Figure 5.49. The derived tree pair and derivation tree for sentence (45) are given in Figure 5.50. The undesired reading is prevented because *himself* is bound by the lambda term to which *mary* is an argument. We discard this bound reading because of the gender mismatch between *mary* and *himself*.[13]

There are two acceptable readings for sentence (46), one in which *him* is coindexed with *john* and one in which the pronoun remains free. Because of the semantics of

[13]No reading where *himself* is coindexed with *John* is available because the extrapolation of the embed features introduced earlier in the chapter prevent *thinks* from adjoining at link 4b when *mary* adjoins at link 1.



⟨ John thinks Mary likes himself.
$\lambda.thinks((\lambda.(\lambda.likes(1))(1)(1))(mary))(1))(john)$
$\lambda x.thinks((\lambda y.(\lambda z.likes(z))(y)(y))(mary))(x))(john)$
$thinks(likes(mary)(mary))(john)$



⟨ John thinks Mary likes him.
$\lambda.thinks((\lambda.(\lambda.likes(1))(>1)(1))(mary))(1))(john)$
$\lambda x.thinks((\lambda y.(\lambda z.likes(z))(x)(y))(mary))(x))(john)$
$thinks(likes(john)(mary))(john)$
or
$\lambda x.thinks((\lambda y.(\lambda z.likes(z))(w)(y))(mary))(x))(john)$
$thinks(likes(w)(mary))(john)$

Figure 5.50: The derived semantics and derivation tree for the sentences "John thinks Mary likes himself" and "John thinks Mary likes him".

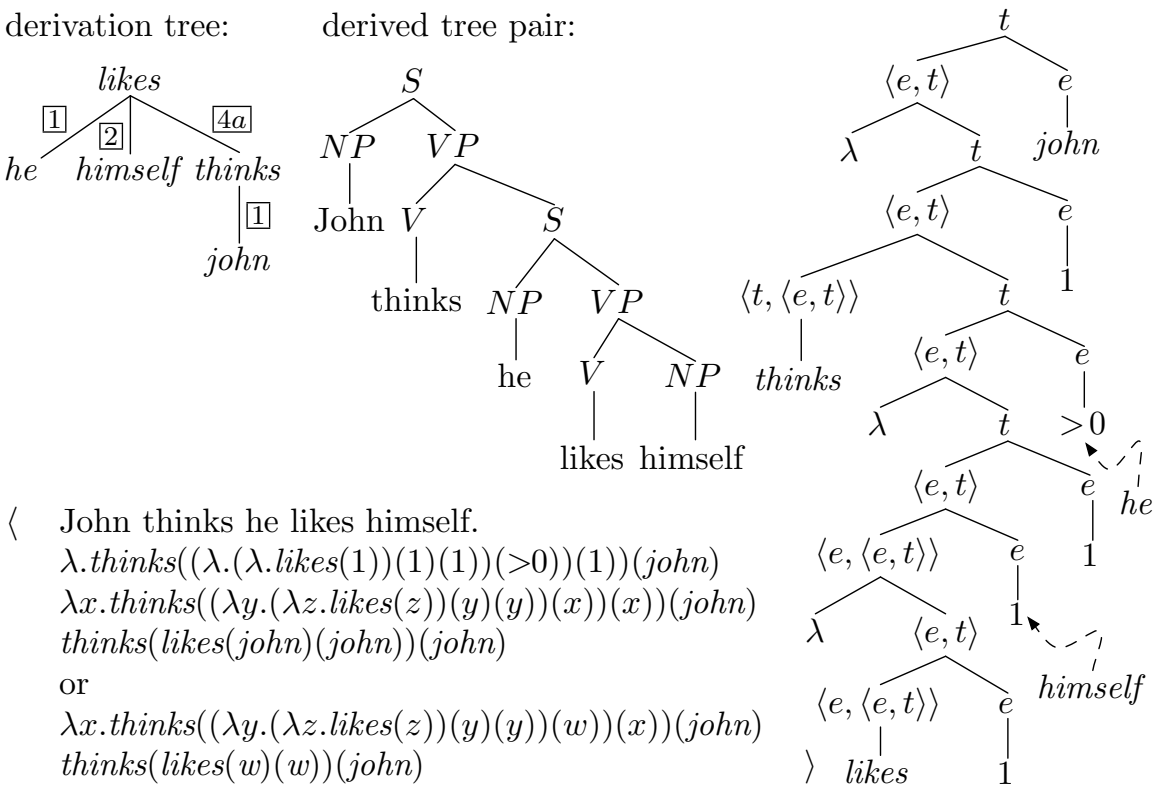


Figure 5.51: The derived tree pair and derivation tree for the sentence "John thinks he likes himself."

him in our grammar, we produce both readings. If *him* takes on the value 2, we get the bound reading. If it takes on a higher value, we produce the free reading. The derivation tree and derived semantics are shown in Figure 5.50.

For sentence (47) there are also two acceptable readings, both shown in Figure 5.51. In both readings, *himself* is coindexed with *he*. However, there is one reading in which *he* is coindexed with *john* and one in which *he* remains free. If *he* takes on the value 1, we produce the reading in which *john*, *he* and *himself* are all coindexed. If *he* takes on a higher value, we produce a reading in which *he* and *himself* are coindexed with each other but not with *john*.

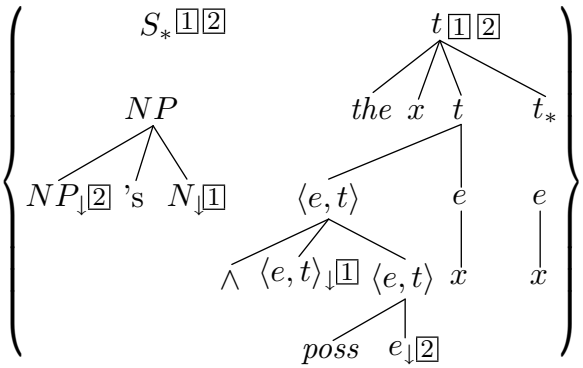


Figure 5.52: A first attempt at a lexical entry for *'s*.

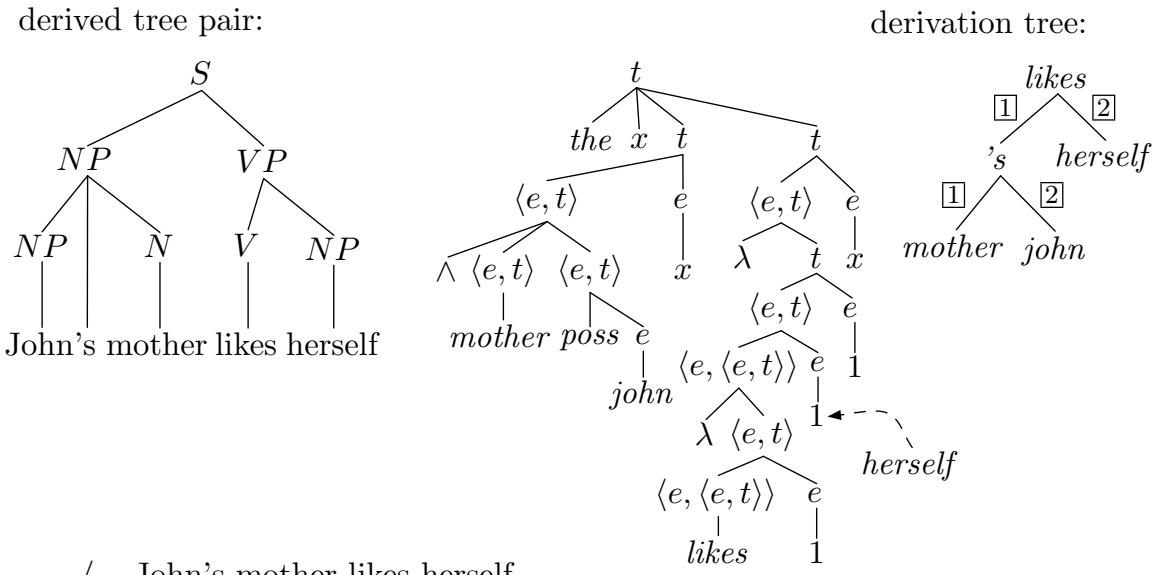
Possessives present another interesting context because both the possessor and the possessed can be coindexed with a pronoun. Consider the following sentences:

(48) John's mother$_i$ likes herself$_i$.

(49) *John$_i$'s mother likes himself$_i$.

(50) John$_i$'s mother likes him$_i$.

Sentence (50) can be analyzed with a treatment of possessives that structures them similar to quantifiers. This first pass analysis of possessives is given in Figure 5.52. Using this lexical entry we produce the derived tree pair and derivation tree for sentence (48) given in Figure 5.53. It is easy to see that sentence (49) will never be derivable because the reflexive pronoun will always coindex with the subject of the sentence. However, sentence (50) poses a conundrum because it is possible for the pronoun to "reach in" to the possessive to coindex with *john*. Based on our current lexical entry for possessives this is not possible because *john* does not appear as the argument to a λ and further, even if it did, that λ would not be a binder for a de



⟨ John's mother likes herself.
$the(x, mother(x) \wedge poss(john)(x), (\lambda.(\lambda.likes(1))(1)(1))(x))$
$the(x, mother(x) \wedge poss(john)(x), (\lambda y.(\lambda z.likes(z))(y)(y))(x))$
$the(x, mother(x) \wedge poss(john)(x), likes(x)(x))$ ⟩

Figure 5.53: Derived tree pair and derivation tree for the sentence "John's mother likes herself."

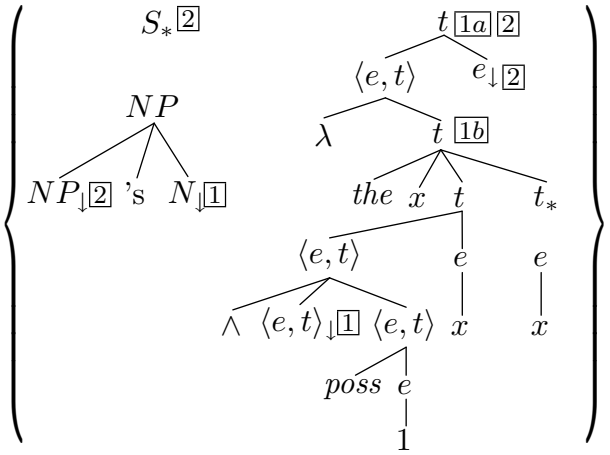
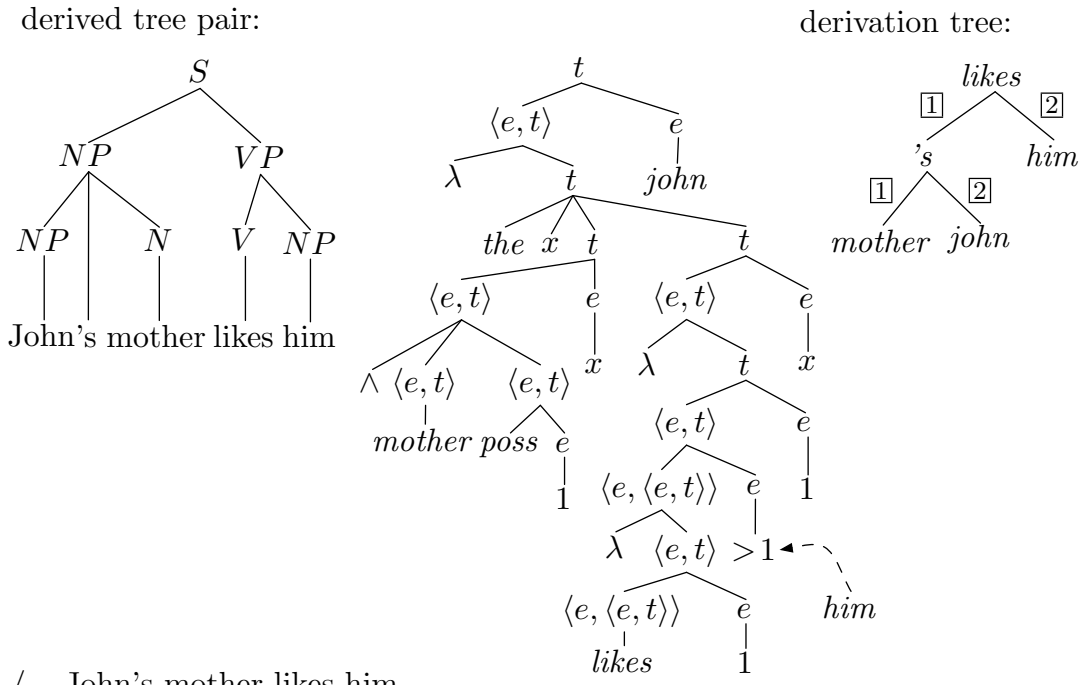


Figure 5.54: A revised lexical entry for *'s* that exposes the entity argument so that it can be coindexed with a pronoun.



⟨ John's mother likes him.
$(\lambda.the(x, mother(x) \wedge poss(1)(x), (\lambda.(\lambda.likes(1))(>1)(1))(x)))(john)$
$(\lambda w.the(x, mother(x) \wedge poss(w)(x), (\lambda y.(\lambda z.likes(z))(w)(y))(x)))(john)$
$the(x, mother(x) \wedge poss(john)(x), likes(john)(x))$ ⟩

Figure 5.55: Derived tree pair and derivation tree for the sentence "John's mother likes him."

Bruin index in the object entity position. An updated lexical entry for *'s*, given in Figure 5.54 remedies this problem by exposing its internal entity argument and making it the argument of a λ in much the same way entity arguments are made available in the lexical entries for verbs.

Using this new lexical entry we are able to analyze sentence (50) as shown by the derived tree pair and derivation tree given in Figure 5.55.[14] In addition, we also produce the correct readings for sentences using quantifiers in the possessive, such as:

[14]Note also that the analysis for sentence (48) continues to be correct with the updated lexical entry for *'s*.

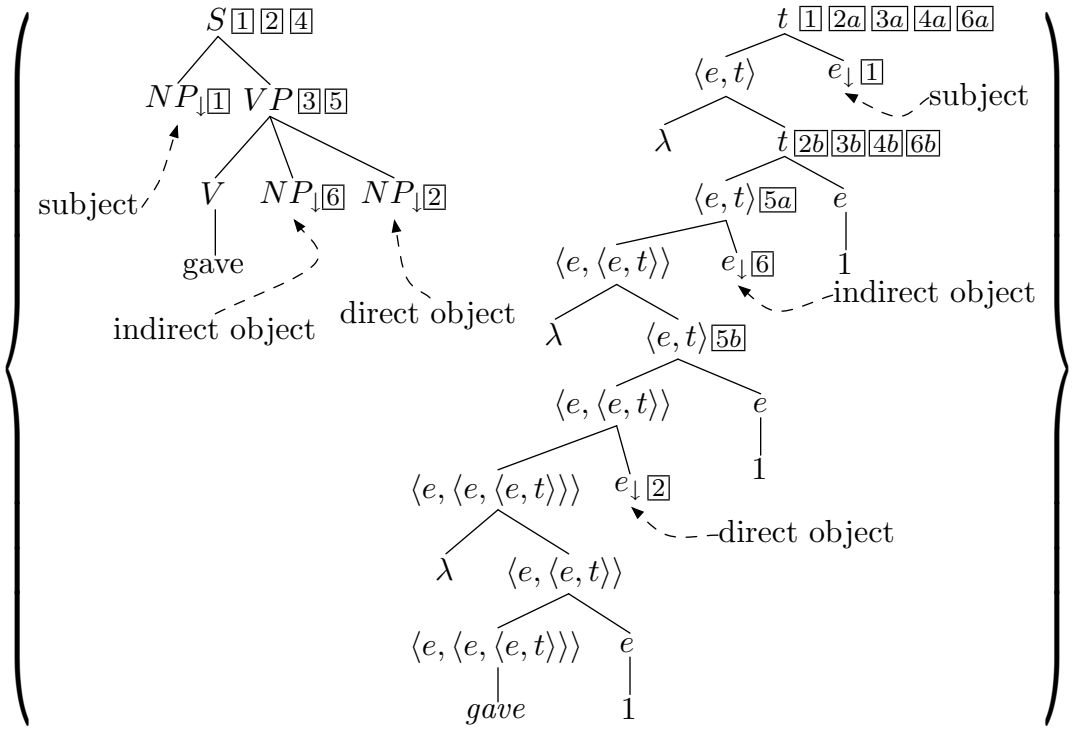


Figure 5.56: The lexical entry for *gave*.

(51) Everyone's mother likes him.

Ditransitive verbs also present a tricky problem because they give rise to cases in which the distribution of reflexive and non-reflexive pronouns overlap. In many of these cases, one reading is preferred but the other reading is still possible given an appropriate context. Consider the following sentences.

(52) John gave Bill$_i$ himself$_i$.

(53) John$_i$ gave Bill himself$_i$.

(54) John$_i$ gave Bill two books about himself$_i$.

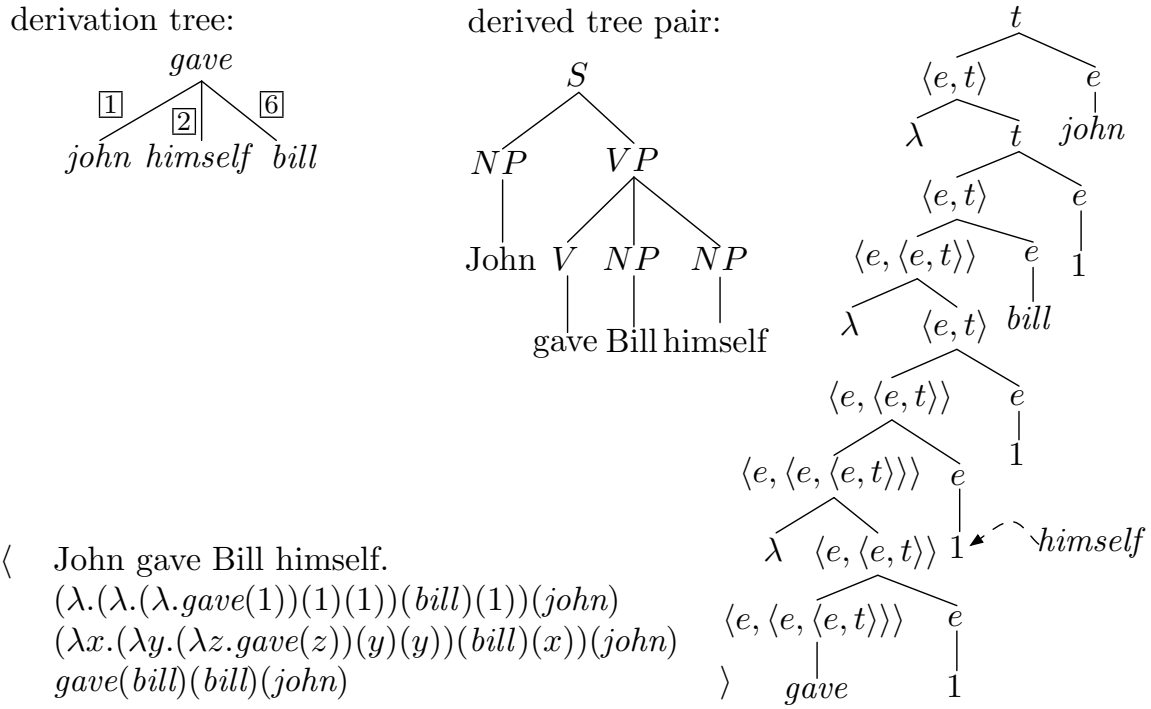


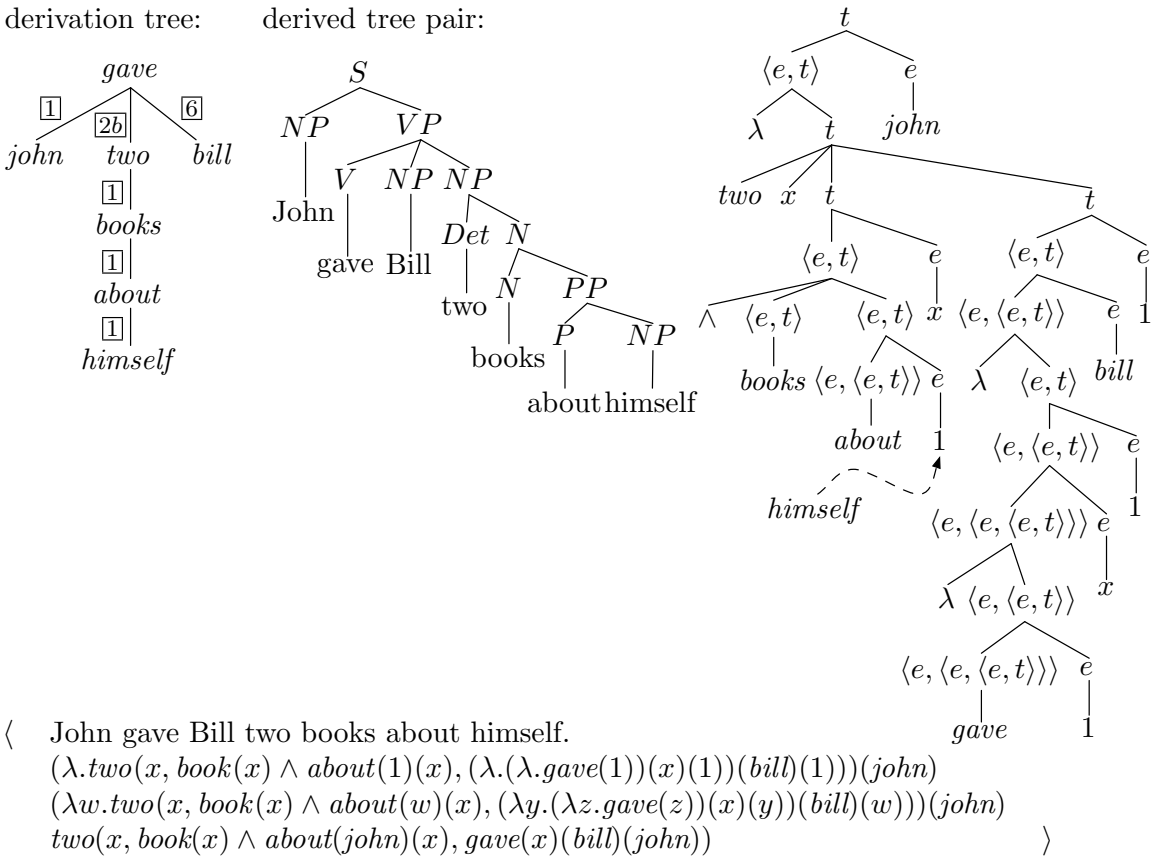
Figure 5.57: Derived tree pair and derivation tree for the sentence "John gave Bill himself" producing only the reading in which *bill* is coindexed with *himself*.

(55) John gave Bill$_i$ two books about himself$_i$.

(56) John gave Bill$_i$ two books about him$_i$.

To analyze these sentences, a lexical entry for a ditransitive verb such as *gave* is needed. The structure of the lexical entries for transitive verbs suggests what it might be (Figure 5.56), and this structure allows generation of some the above sentences.

Consider sentences (52) and (53). The sentence itself is odd, making it difficult to have a strong intuition about which reading is preferred. However, it appears that both the reading in which *himself* coindexes with *bill* and the one in which it coindexes with *john* are available. Figure 5.57 demonstrates the only analysis available



⟨ John gave Bill two books about himself.
$(\lambda.two(x, book(x) \wedge about(1)(x), (\lambda.(\lambda.gave(1))(x)(1))(bill)(1)))(john)$
$(\lambda w.two(x, book(x) \wedge about(w)(x), (\lambda y.(\lambda z.gave(z))(x)(y))(bill)(w)))(john)$
$two(x, book(x) \wedge about(john)(x), gave(x)(bill)(john))$ ⟩

Figure 5.58: Derived tree pair and derivation tree for the sentence "John gave bill two books about himself" producing only the reading in which *himself* is coindexed with *john*.

with the lexical entry for *gave*, which produces the reading from sentence (52). Because the indirect object intervenes between the subject and the direct object in the semantic tree, this analysis does not produce the arguably acceptable reading from sentence (53).

Next consider sentences (54) and (55). Again, both readings are available, but with these sentences the contrast is clear. The reading in which *himself* is coindexed with *john* is much preferable to the one in which it is coindexed with *bill*. As shown in

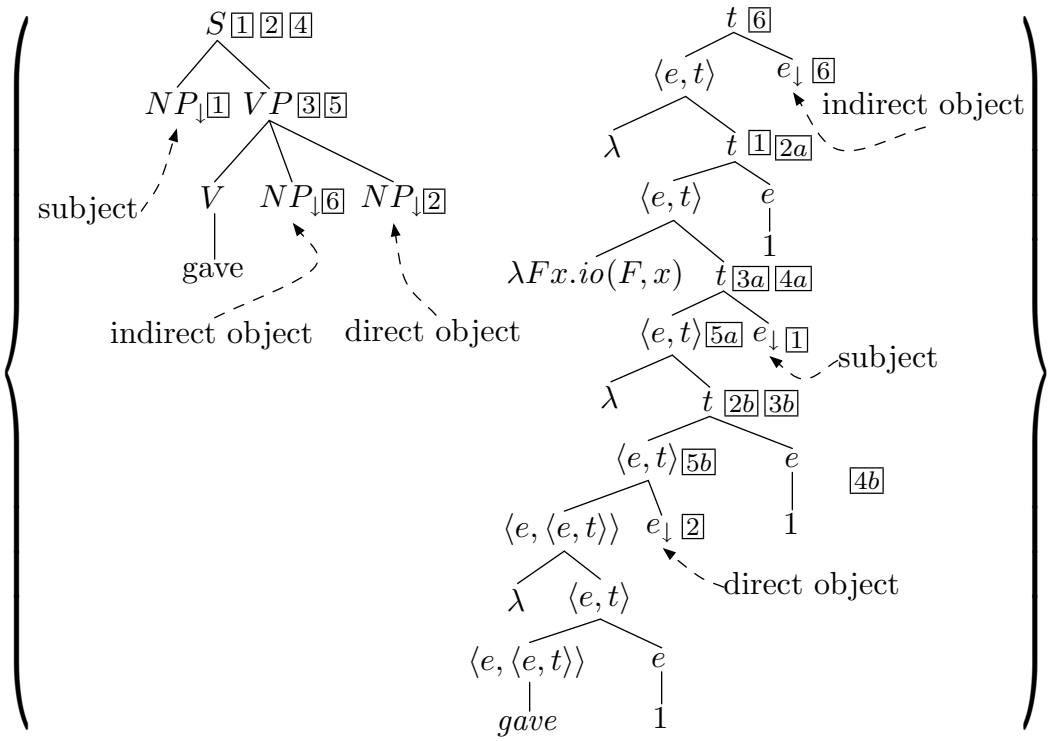
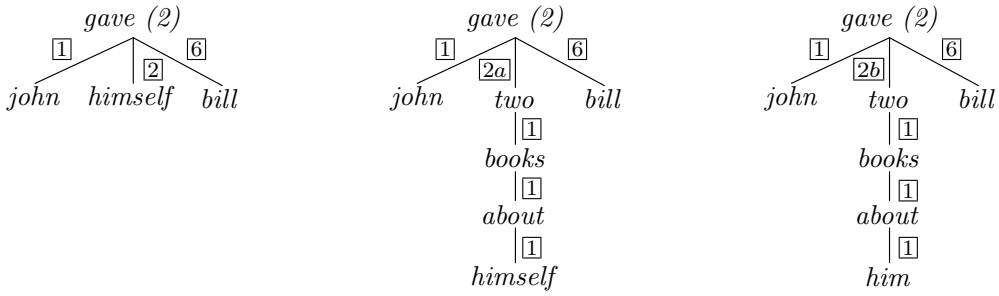


Figure 5.59: An additional lexical item for *gave*.

Figure 5.58, the preferred reading is the one produced by the current lexicon. However, the reading in which *himself* is coindexed with *bill* is unavailable. In addition, sentence (56) cannot be produced from the lexicon (although the arguably preferred unbound reading of *him* is available).

To analyze the sentences that are not captured by the current lexical entry for *gave* one can observe that what is needed is a treatment of ditransitives where the indirect object is the outermost argument to the verb. It is possible to generate an alternative lexical item for *gave* that achieves this by analyzing the indirect object with a relation that holds between the proposition generated by a transitive version of *gave* and the indirect object entity, as is shown in Figure 5.59. This additional



⟨ John gave Bill himself.
$(\lambda.(\lambda Fx.io(F,x))((\lambda.(\lambda.gave(1))(1)(1))(john))(1))(bill)$
$(\lambda w.(\lambda Fx.io(F,x))((\lambda y.(\lambda z.gave(z))(y)(y))(john))(w))(bill)$
$io(gave(john,john),bill)$ ⟩

⟨ John gave Bill two books about himself.
$(\lambda.two(x,book(x) \wedge about(1)(x),(\lambda Fx.io(F,x))((\lambda.(\lambda.gave(1))(x)(1))(john))(1)))(bill)$
$(\lambda w.two(x,book(x) \wedge about(w)(x),(\lambda Fx.io(F,x))((\lambda y.(\lambda z.gave(z))(x)(y))(john))(w))(bill)$
$two(x,book(x) \wedge about(bill)(x),io(gave(x)(john),bill))$ ⟩

⟨ John gave Bill two books about him.
$(\lambda.(\lambda Fx.io(F,x))((\lambda.two(x,book(x) \wedge about(>1)(x),(\lambda.gave(1))(x)(1)))(john))(1)(bill)$
$(\lambda w.(\lambda Fx.io(F,x))((\lambda y.two(x,book(x) \wedge about(w)(x),(\lambda z.gave(z))(x)(y)))(john))(w)(bill)$
$io(two(x,book(x) \wedge about(bill)(x),gave(x)(john)),bill)$ ⟩

Figure 5.60: Derivation trees using the additional lexical entry for *gave* and the readings that they produce.

lexical entry generates readings for the remaining example sentences. The derivation trees for sentences (53), (55), and (56) are given in Figure 5.60.

Finally, we examine the issue of crossover. In linguistics, crossover refers to phenomena in which a potential binder, such as a Wh-word or a quantifier moves across a pronomial. Strong crossover refers to those cases in which the pronomial c-commands the extraction site in the syntax and weak crossover to those in which it does not.

A canonical example of strong crossover is given in Sentence (57).

(57) *Who$_i$ does he$_i$ like t_i?

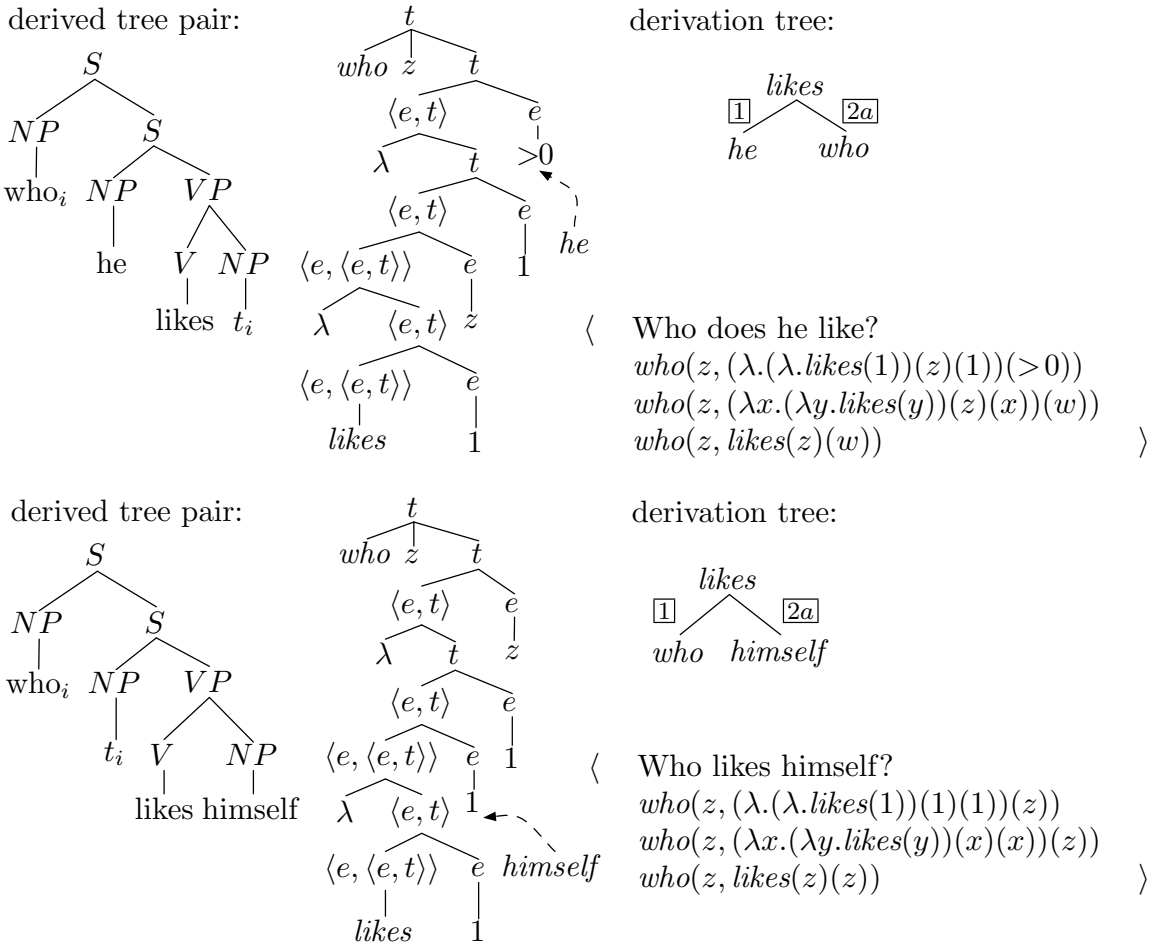


Figure 5.61: At the top, the derived tree pair and derivation tree for the sentence "Who does he like?", demonstrating the impossibility of coindexing *who* with *he*. Below, the derived tree pair and derivation tree for the sentence "Who likes himself?"

The ungrammaticality of this coindexation is predicted because the only way that the Wh-word and the pronoun could be coindexed is if the bound variable for the Wh-word is the argument to the lambda that binds the pronoun. Because the pronoun is the subject and the bound variable of the Wh-word is the object, resulting in the pronoun appearing higher in the semantic tree than the bound variable of the Wh-word, this cannot happen. This orientation of the pronoun and Wh-word bound variable is directly related to the c-command relationship in the syntax, although the

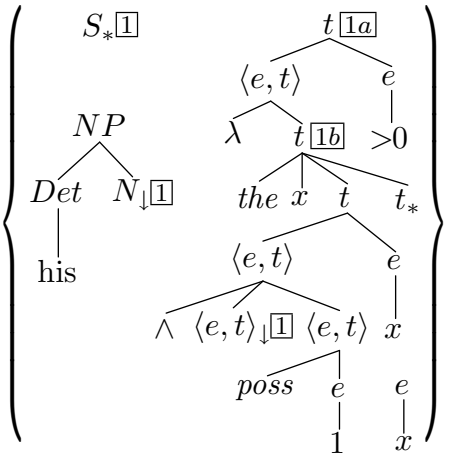


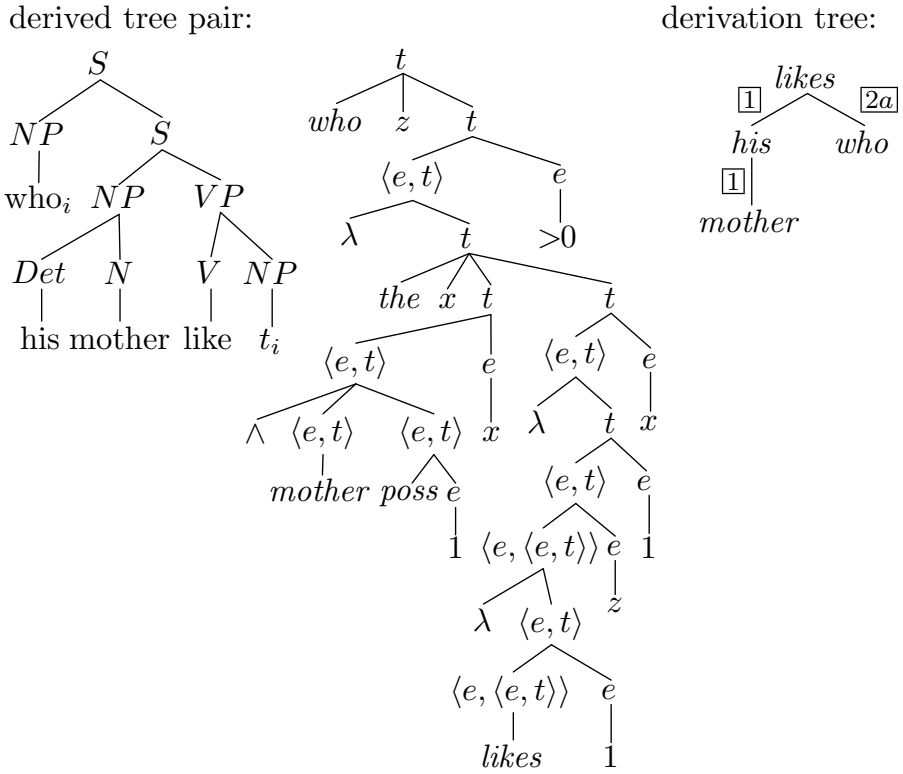
Figure 5.62: The lexical entry for *his*.

relationship can be directly stated in the semantics in this system. The analysis of this sentence is given in Figure 5.61. Contrast it with the successful coindexation of *who* and *himself* in the following sentence (also given in Figure 5.61):

(58) Who$_i$ t_i likes himself$_i$?

The analysis of weak crossover effects falls out directly from the semantics of pronomial constructions such as *his mother* (Figure 5.62). Because possessives are analyzed like quantifiers, they attach at t. If the pronomial is in the subject of the sentence, it is forced to attach at the highest t node (link 1), which prevents it from being bound by a binder within the same clause. Consider the example given in Sentence (59) and the analysis of it in Figure 5.63. Regardless of the order in which *who* and *his* take scope, the bound reading of *his mother* cannot be produced.

(59) *Who$_i$ does his$_i$ mother like t_i?

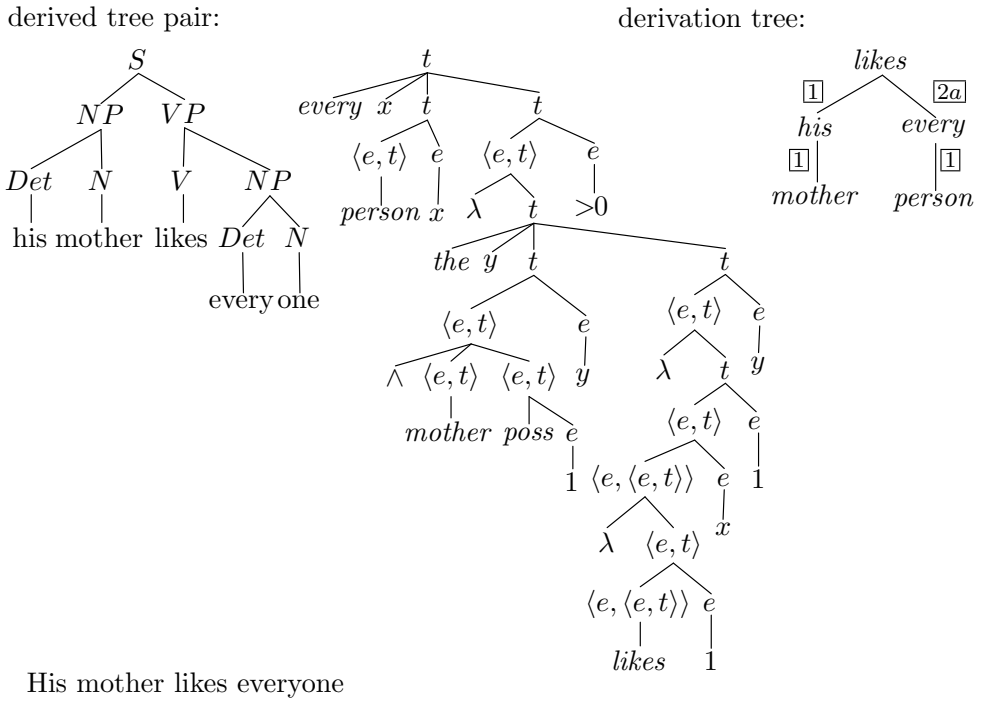


⟨ Who does his mother like?
$who(z, (\lambda.the(x, mother(x) \wedge poss(1)(x), (\lambda.(\lambda.likes(1))(z)(1))(x)))(>0))$
$who(z, (\lambda w.the(x, mother(x) \wedge poss(w)(x), (\lambda y.(\lambda v.likes(v))(z)(y))(x)))(u))$
$who(z, the(x, mother(x) \wedge poss(u)(x), likes(z)(x)))$ ⟩

Figure 5.63: Derived tree pair and derivation tree for the sentence "Who does his mother like?". Although only one derived semantics is shown here, the bound reading of *his mother* cannot be produced even if *his* outscopes *who*.

The grammar also correctly predicts the ungrammaticality of sentence (60) under the bound reading. In this case, although it is possible for the scope part *everyone* to attach higher than *his mother*, because *his mother* is in the subject position, its attachment point must be higher than the bound variable of *everyone*, preventing coindexation. The analysis is shown in Figure 5.64.

(60) *His$_i$ mother likes everyone$_i$.



⟨ His mother likes everyone
every$(x, person(x), (\lambda.the(y, mother(y) \wedge poss(1)(y), (\lambda.(\lambda.likes(1))(x)(1))(y)))(>0))$
every$(x, person(x), (\lambda w.the(y, mother(y) \wedge poss(w)(y), (\lambda z.(\lambda v.likes(v))(x)(z))(y)))(u))$
every$(x, person(x), the(y, mother(y) \wedge poss(u)(y), likes(x)(y)))$ ⟩

Figure 5.64: The derived tree pair and derivation tree for the sentence "His mother likes everyone."

5.11 Conclusion

This chapter addresses a wide range of constructions of English using synchronous TL-MCTAG to model the interface between syntax and semantics. The fragment of English covered includes phenomena in which the syntactic and semantic structures are so divergent that many thought a synchronous TAG analysis would not be possible. In particular, cases of long distance movement, scope ambiguity, and pronoun binding that have been considered hard are successfully addressed here in novel ways. However, this chapter also pushes the boundaries of synchronous TL-MCTAG and

demonstrates that there are certain constructions of English that cannot readily be analyzed within this formalism. These include nested quantifiers, the interaction of control verbs with quantifiers and other scope-taking elements, and certain sentences in which quantifiers take scope above the relative clause in which they appear. Analyses of these hard cases are the subject of the following chapter.

Chapter 6

Modeling the Syntax-Semantics Interface: Beyond TL-MCTAG

6.1 Introduction

Certain phenomena and constructions cannot be modeled with TL-MCTAG alone as the base formalism. In this chapter we explore these cases and how they can be handled using an extension to TL-MCTAG introduced in Chapter 3. The challenges of these cases uniformly arise from situations in which movement beyond the usual locality boundaries appears to occur. These cases include constraints on quantificational elements that attach along the VP-spine, nested quantifiers or inverse linking, and escape of quantifiers from islands. We use a limited extension to TL-MCTAG, Limited Delay V-TAG (LDV-TAG), to analyze these cases without substantially increasing the complexity of the base formalism.

6.1.1 Introducing Dominance and Delay

Although the locality constraints of TL-MCTAG naturally produce appealing predictions about locality and movement in most of the cases we have examined, in a few examples scope-taking elements appear to be able to take scope higher in the derived tree than synchronous TL-MCTAG permits. We return to the concept of limited delay, introduced in Chapter 3, in order to permit analyses of these cases without losing the constraints imposed by TL-MCTAG locality or the processing advantage of TL-MCTAG over SL-MCTAG and non-local MCTAG. We do this by changing our base formalism from TL-MCTAG to LDV-TAG. LDV-TAG allows us to maintain the efficiency of TL-MCTAG while also taking advantage of certain characteristics of our lexicon and providing the flexibility needed for those cases where TL-MCTAG constrains locality too much.

We begin by noting that in all of the multicomponent tree sets we have introduced, and particularly in those that model scope-taking elements we have assumed an implicit dominance relation between the trees in the set. In the scope-taking elements we have assumed that the scope tree must dominate the variable tree in the derived semantics. We now make these relations explicit by adding dominance links to our lexicon. This change is required by the LDV-TAG formalism and helps to constrain its complexity by allowing the parser to keep track of only tree in a set at a time as it processes the derived tree bottom-up.

We also add delayed trees to some of our tree sets. Recall the highest tree in the dominance chain of an LDV-TAG tree set may be marked with a diacritic ($\uparrow$) to indicate that it may adjoin in a different tree than the other trees in its tree set

so long as it obeys its dominance requirement. Noting that the cases in which more flexibility in movement is required are those in which we need a scope-taking element to take scope higher than its TL-MCTAG domain of locality, we employ this delay for trees that represent scope takers. Although it is not part of the LDV-TAG formalism, we employ a principle of preference for least delay. Although we permit delayed trees to rise to any point in the derived tree, we expect that readings that use no delay or less delay will generally be preferred to those that employ long delays.

6.2 Nested Quantifiers

Consider the sentence:

(1) Someone likes every character in two books.

Although a nested quantifier may take scope over the quantifier within which it is nested (so-called "inverse linking") not all permutations of scope orderings of the quantifiers are available [Joshi et al., 2003]. In particular, at least one and perhaps both readings in which a quantifier intervenes between a nesting quantifier and its nested quantifier are not valid [VanLehn, 1978, Hobbs and Shieber, 1987]. In the example sentence (1), this predicts that the reading *two* > *some* > *every* should under no circumstances be available and, according to some, that the reading *every* > *some* > *two* should also not be valid.

The lexicon introduced Chapter 5 (with minor modification) produces the four widely accepted readings for this sentence. The key lexical items used in this derivation are shown in Figure 6.1. Only the lexical item for quantifiers is changed in that

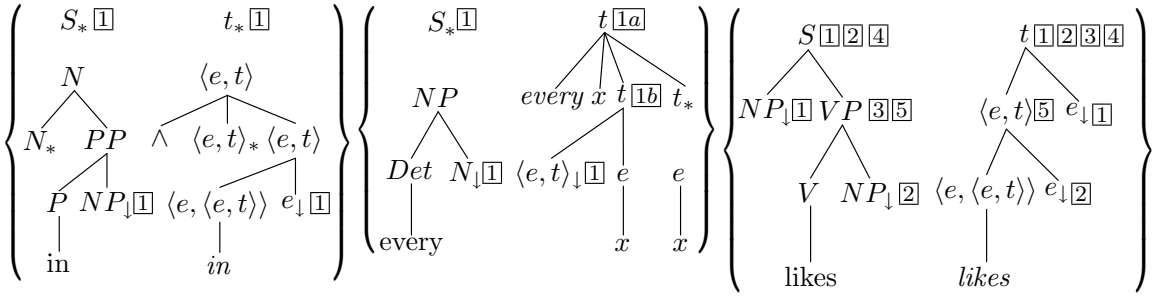


Figure 6.1: Representative lexical entries for a set-local analysis of the sentence "Someone likes every character in two books." The lexical entry for the quantifier is updated by splitting the adjunction site for the scope part of noun phrase meanings into two locations (1a and 1b). The lexical entries for *in* and *likes* are provided for convenience.

it now has two locations for the scope part to attach in the semantics. The derived trees and derived semantics for sentence (1) are given in Figure 6.2. In this analysis, because the nested quantifier is introduced through the prepositional phrase, which in turn modifies the noun phrase containing the nesting quantifier, the two quantifiers already naturally form a set that operates as a unit with respect to the rest of the derivation. The four readings are produced by the combination of the alternative locations for the adjunction of the noun into the nesting quantifier and the multiple adjunction of the nesting quantifier and the third quantifier at the root of the verb. However, a serious problem with this analysis is that it is set-local rather than tree-local. That is, the lexical entries for prepositions and nouns contain links that have locations in more than one tree in the set. As discussed in the first part of this thesis, the computational ramifications of this are unacceptable. In addition, even with the move to set locality, the fifth arguably acceptable reading cannot be derived.

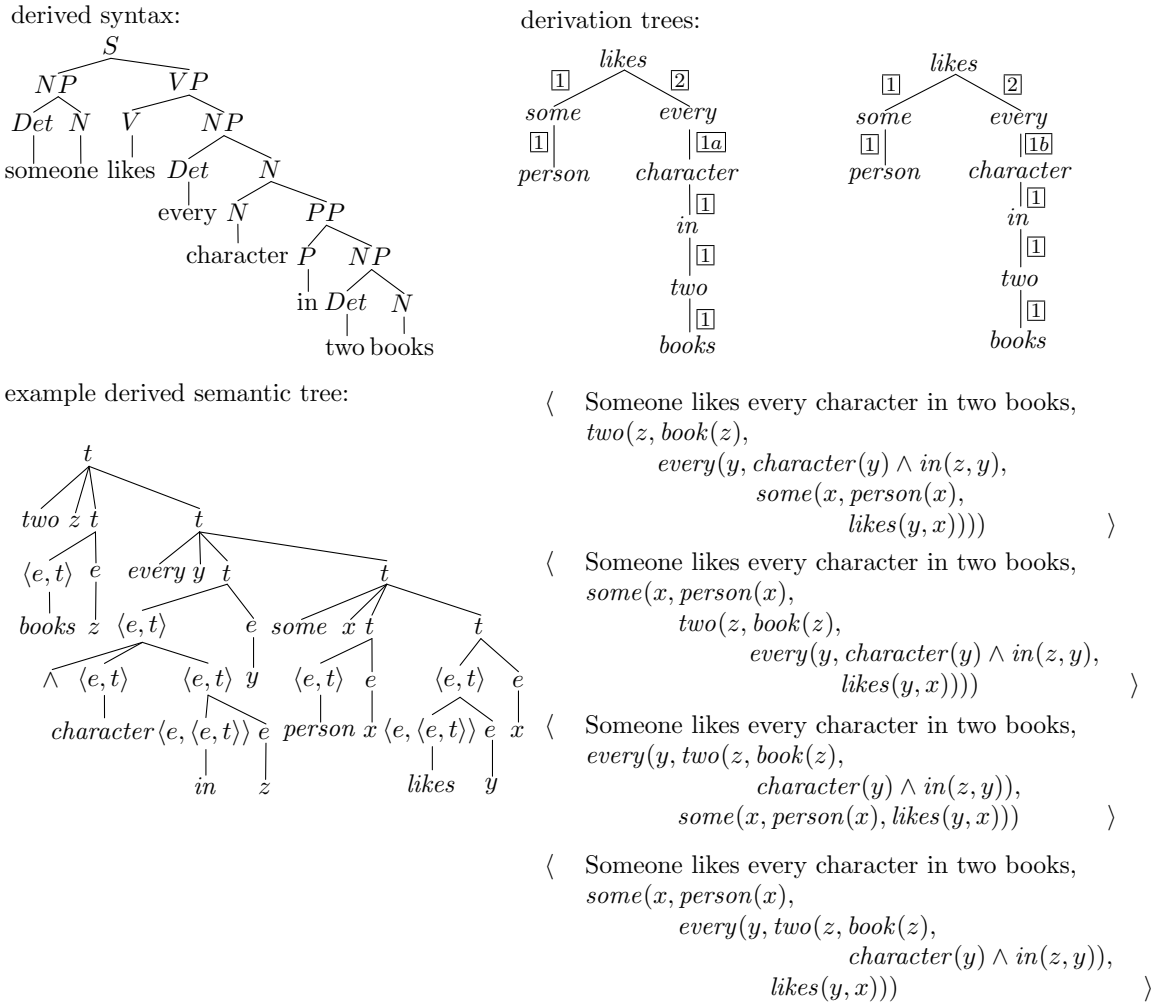


Figure 6.2: Derived trees and derivation trees the sentence "Someone likes every character in two books." Only a single derived semantic tree is given as an example. The derived semantics for all four readings are provided. Note that each derivation tree corresponds to two derived semantic trees.

Using delay, the original structure for prepositions no longer poses a problem or requires the proliferation of vestigial t_* trees throughout the grammar. Figure 6.3 shows the lexical entries under an LDV-TAG analysis. Because there is no adjunction site for the scope part of the object of a preposition in the semantic trees of the preposition or noun, the scope part must delay and attach higher. There are several

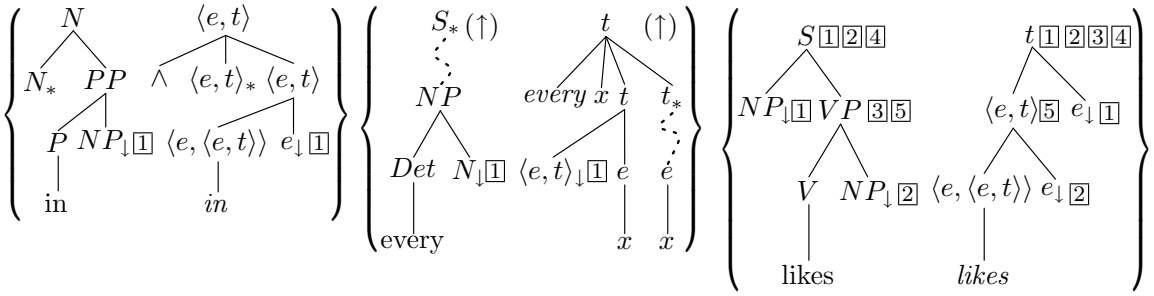
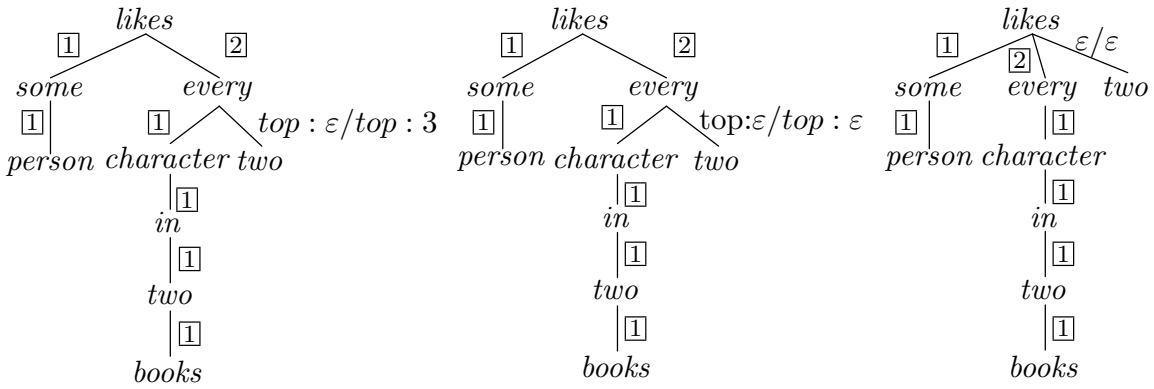


Figure 6.3: Lexical entries for a delay-based analysis of the sentence "Someone likes every character in two books."

possible adjunction sites where it may attach, which correspond to the desired five readings of the example sentence. This results in the possible derivations for sentence (1), shown in Figure 6.4. The four readings in which *some* does not intervene between *two* and *every* are produced by delaying the attachment of the scope part of *two* until one of the attachment points in the *every* tree. That is, because there is no location for the scope part of *two* to adjoin in *character* or *in*, it must delay higher in the derived tree. The next t node on the path up the derived tree is the lower t node in the *every* tree. If *two* attaches at this node, *every* and *two* will become indivisibly attached in the *every* > *two* ordering. Together, they will multiply adjoin with *some* at the root of the *likes* tree to produce the scope orderings *every* > *two* > *some* and *some* > *every* > *two*. If the scope part of *two* attaches at the higher t node in the *every* tree, it will become indivisibly attached in the *two* > *every* ordering. Once again, they will multiply adjoin with *some* at the root of the *likes* tree to produce the scope orderings *two* > *every* > *some* and *some* > *two* > *every*. The fifth reading is produced if the scope part of *two* delays until after *every* has adjoined to the *likes* tree. At that point, *two* may adjoin above *every* at the root of the *likes* tree. This permits



⟨ Someone likes every character in two books,
two(*z*, *book*(*z*),
some(*x*, *person*(*x*),
every(*y*, *character*(*y*) ∧ *in*(*z*, *y*)),
likes(*y*, *x*))) ⟩

Figure 6.4: The three derivation trees produced for the sentence "Someone likes every character in two books." as well as the fifth semantic reading that is derived. The other four semantic readings are the same as those produced by the non-delay lexicon. The locations at which the delayed trees adjoin are labeled in the derivation tree with their Gorn addresses in the syntax and semantics because the adjunctions do not occurring at links.

the scope ordering *two* > *some* > *every*. The reading in which *every* outscopes *two* with *some* intervening is unavailable.

It is interesting to note that the disputed fifth reading requires the scope part of *two* to delay further both as measured in the derived tree and in the derivation tree. Although it is not clear how to block this reading entirely without something like a hard bound on the degree of delay, the principle of least delay does provide some justification for it being dispreferred with respect to the four other readings produced.

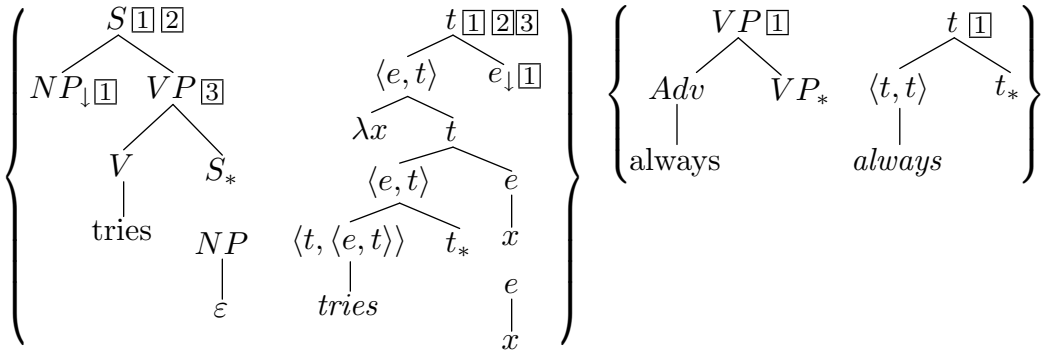


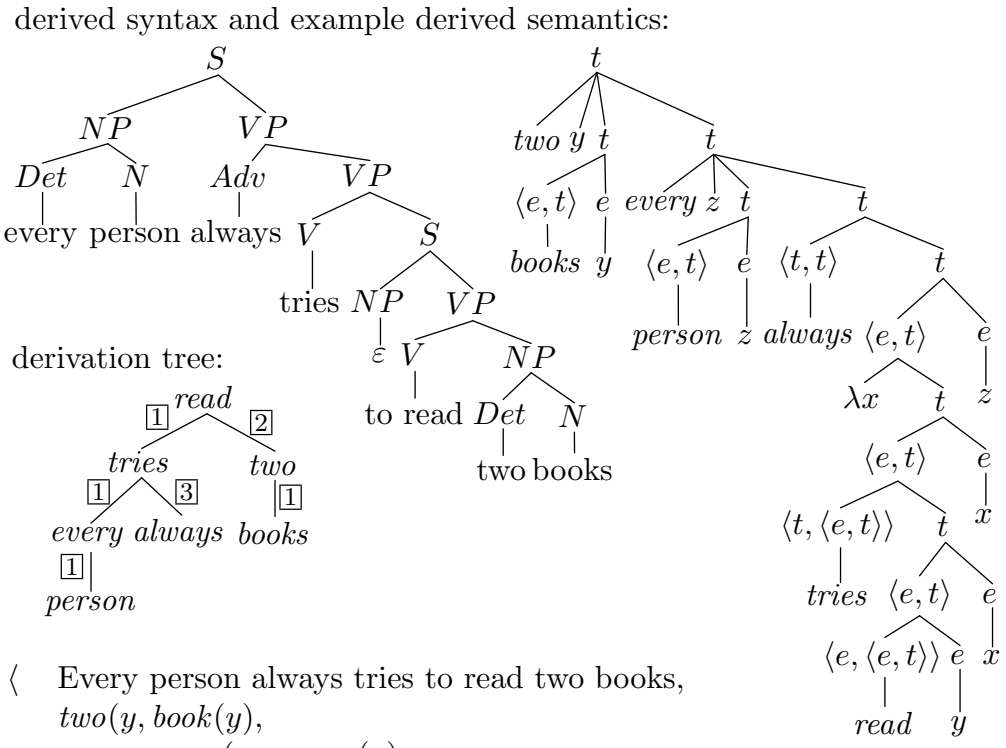
Figure 6.5: The lexical items for an initial analysis of the sentence "Every person always tries to read two books." The structure of *read* is assumed to be the same as *likes*.

6.3 Quantificational Elements Along the VP-Spine

It is widely accepted that quantificational elements take scope in the order in which they attach along the VP spine. However, they appear to scope freely with respect to quantificational elements that do not attach along the VP spine. This poses problems for the analysis of adverbs, which are represented as single tree sets that take scope where they attach. Consider the following sentence:

(2) Every person always tries to read two books.

In this sentence *always* and *tries* both attach along the VP spine and must take scope in the order in which they appear. However, *two* is free to scope in any order with respect to *always* and *tries*, and *every* is free to scope anywhere above *tries* because its bound variable is in the subject position of *tries*. An intuitive structure for this sentence and lexical items based on those introduced in the previous chapter are given in Figure 6.5.



⟨ Every person always tries to read two books,
two(*y*, *book*(*y*),
every(*z*, *person*(*z*),
always(*tries*(*read*(*z*, *y*), *z*)))) ⟩

⟨ Every person always tries to read two books,
every(*z*, *person*(*z*),
always(*tries*(*two*(*y*, *book*(*y*),
read(*z*, *y*), *z*)))) ⟩

⟨ Every person always tries to read two books,
two(*y*, *book*(*y*),
always(*every*(*z*, *person*(*z*),
tries(*read*(*z*, *y*), *z*)))) ⟩

⟨ Every person always tries to read two books,
always(*every*(*z*, *person*(*z*),
tries(*two*(*y*, *book*(*y*),
read(*z*, *y*), *z*)))) ⟩

Figure 6.6: The derived syntax and an example derived semantic tree for the sentence "Every person always tries to read two books." The derivation tree and all of the derived semantics are also included.

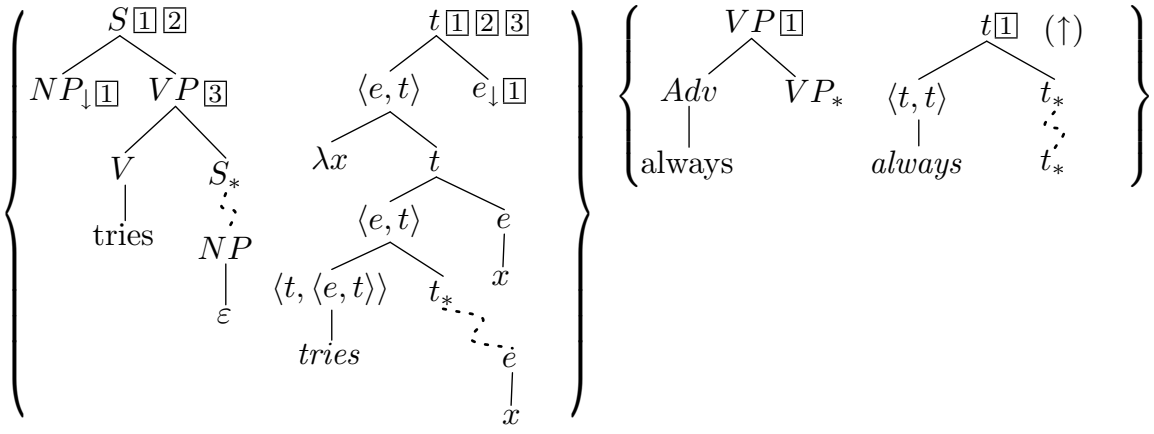
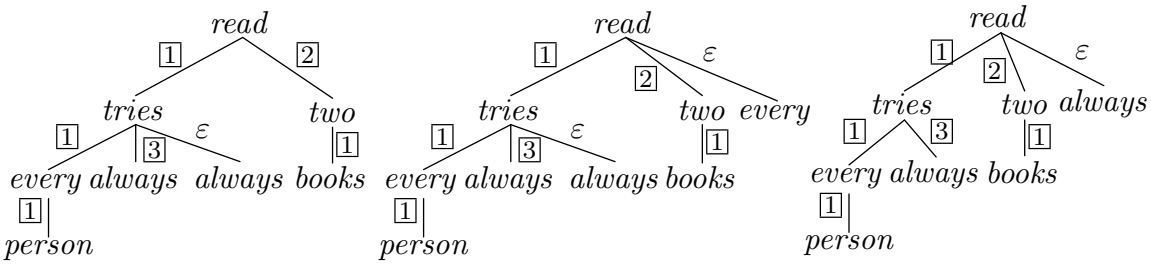


Figure 6.7: The lexical items for the sentence "Every person always tries to read two books." The altered lexical item for *always* permits delay and allows the generation of a sixth reading for the sentence as well as readings in which *two* immediately outscopes *tries* but scopes under *every* and *always*.

The problem with the analysis given in Figure 6.6 is that the scope order of *every*, *always* and *tries* becomes fixed (*every* > *always* > *tries* or *always* > *every* > *tries*) before they adjoin to the *read* tree. This permits four scope readings:

- *every* > *always* > *tries* > *two*
- *two* > *every* > *always* > *tries*
- *always* > *every* > *tries* > *two*
- *two* > *always* > *every* > *tries*

However, we cannot generate the reading *always* > *two* > *every* > *tries*. This is the reading in which at any given time there are two specific books, say the top two books on the New York Times bestseller list, that every person is trying to read. Contrast this with the scope ordering *two* > *always* > *every* > *tries* where the two



⟨ Every person always tries to read two books,
always(*two*(*y*, *book*(*y*),
every(*z*, *person*(*z*),
tries(*read*(*z*, *y*), *z*)))) ⟩

Figure 6.8: The derivation trees for the sentence "Every person always tries to read two books." The derivation tree at the right corresponds to the additional semantic reading shown. The two left hand derivation trees correspond to the semantic readings shown under the previous analysis.

books that everyone is trying to read do not change over time, perhaps challenging classics such as *Swann's Way* and *The Sound and the Fury*. In order to generate this reading, *always* needs to remain unattached until it multiply adjoins with *two* at the root of *read*. For this, we can use delay. Figure 6.7 gives a modified lexicon where the scope of *always* is permitted to delay. Figure 6.8 shows the derivations produced by this altered lexicon.[1]

Allowing the scope part of VP-spine attachers such as adverbs to delay addresses the challenge of this particular example but raises question of how to ensure that VP-spine attachers will remain fixed in the order in which they attach in the syntax. This problem is exemplified by the following sentences:

[1]We also do not obtain certain readings in which *two* immediately outscopes *tries*. These can now be obtained by delaying both *every* and *always* so that all four scope-taking elements can adjoin multiply at the root of *read*. Alternatively (and without use of delay), they can be produced by breaking the *tries* tree into three trees so that the scope parts of *every* and *always* do not attach to the scope part of *tries* until they multiply adjoin at the root of *read*.

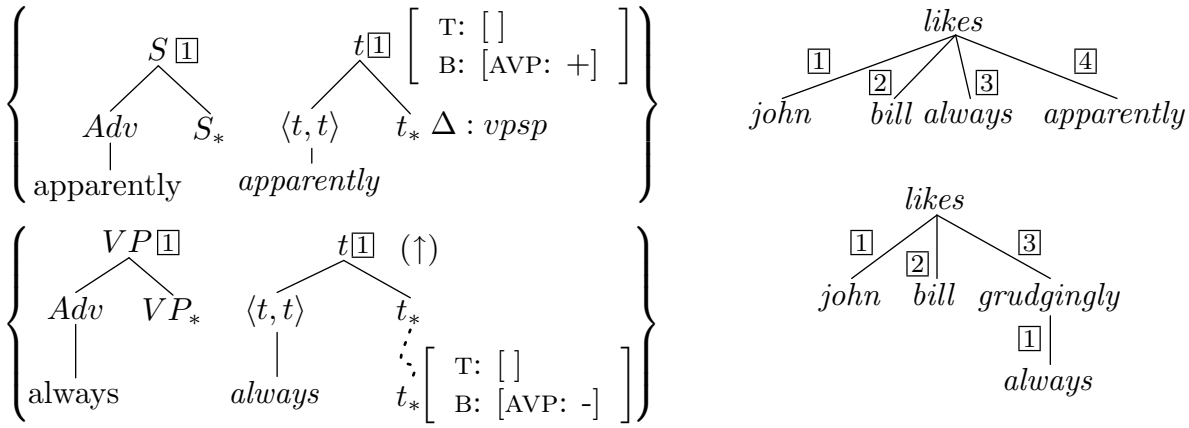


Figure 6.9: The lexical entries for *apparently* and *always* and the derivation trees for the sentences "Apparently John always likes Bill" and "John always grudgingly likes Bill."

(3) Apparently John always likes Bill.

(4) John always grudgingly likes Bill.

In sentence (3), *apparently* must outscope *always*. In sentence (4), *always* must outscope *grudgingly*. In sentence (4), the ordering between *always* and *grudgingly* is maintained because the foundation tree of *always* adjoins at the root of the scope tree of *grudgingly*, so the dominance constraints on *always* guarantee that its scope part will attach higher than that of *grudgingly* (Figure 6.9). However, for sentences like sentence (3), *apparently* must outscope *always* even though they multiply adjoin at the root of *likes*. We use a feature to ensure that the foundation tree of *always* adjoins below *apparently*. Then, in order to block *always* from delaying past *apparently*, we add an integrity constraint to the foot of the semantic tree for *apparently*

that blocks the passage of VP spine attachers.[2] The necessary lexical items and the derivation tree for sentence (3) are given in Figure 6.9.

6.4 Quantifiers Scoping Out of Islands

Although syntactic movement out of islands is prevented by the locality constraints of our grammar in most cases, there are cases of quantifier scope in which we need to allow the scope part of a quantifier to escape the tree set into which it adjoins. For example, consider the sentence:

(5) John saw the person who everyone likes.

There is a strained but arguably available reading for this sentence in which each person likes a different person, implying that *every* outscopes the relative pronoun (and the other quantifier).[3] Because we permit delay of the scope part of generalized quantifiers, this reading is available. However, it requires the scope part of *every* to delay all the way to the root of the scope part of *the*, perhaps explaining why the reading in which *every* remains local to the relative clause is preferred.

The reader may note that this poses a problem for our earlier analysis of movement of Wh words out of subordinate clauses, such as the earlier examples repeated here for convenience:

[2]Because t attaching modifiers such as *apparently* do not delay, there is no need for an integrity constraint on the $\langle e, t\rangle$ attaching modifiers to block their passage. However, if there were reason to allow t attachers to delay, an additional integrity constraint could be added to the foot of the scope tree of the $\langle e, t\rangle$ attachers that blocks passage of VP spine attachers.

[3]If we replace *every* with *each* as in "John saw the boy that each girl likes.", the reading becomes less strained.

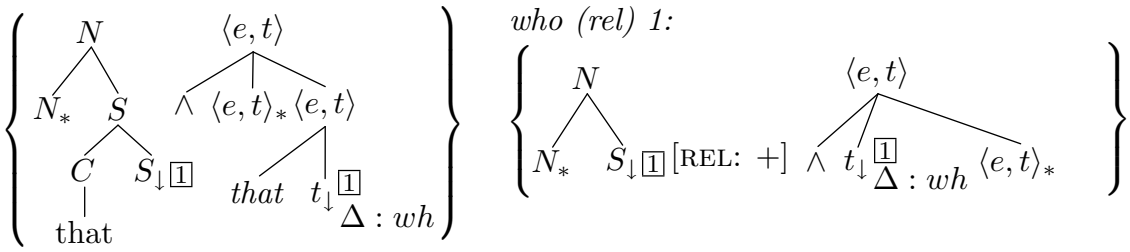
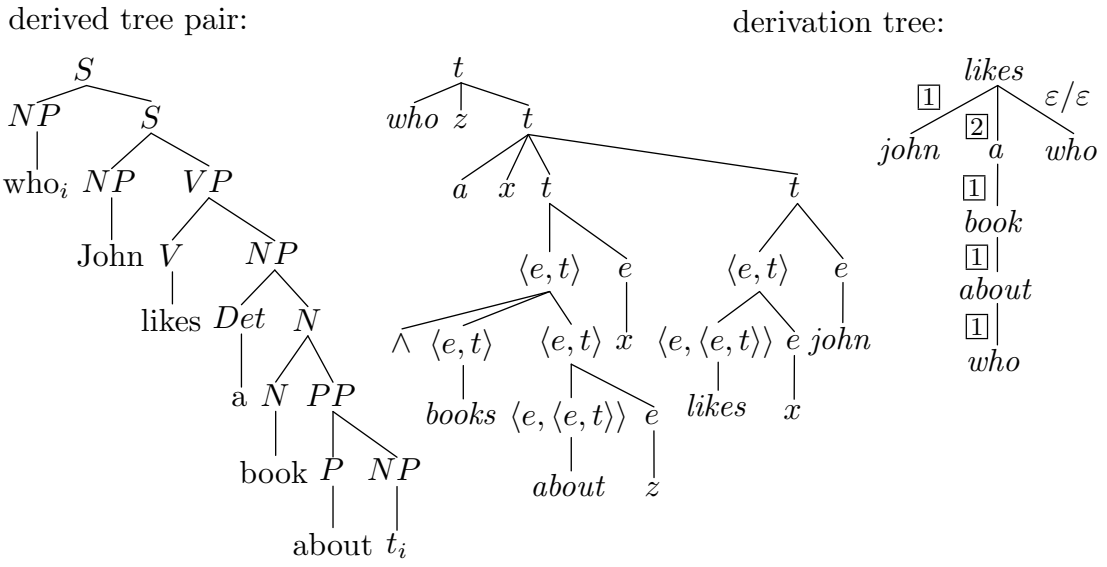


Figure 6.10: Lexical entries for *that* and the relative pronoun *who* updated with integrity constraints that prevent the delay of Wh words beyond their substitution nodes.

(6) *Who$_i$ does John believe the report that Mary likes t_i?

(7) *Who$_j$ does Bill have two brothers$_i$ who$_i$ t_i like t_j?

In order to allow Wh words to be the objects of prepositions (as in sentences like "Who$_i$ does John like a book about t$_i$?"), we have to either create an adjunction site for them in the lexical entries for prepositions and nouns—an alternative we discarded before because it makes our grammar set-local—or we have to allow the scope parts of Wh words (and other noun phrases) to delay. However, we wish to limit the extent to which a Wh word can delay so that we do not generate sentences like sentence (6) and (7). In order to prevent this kind of movement we add an integrity constraint that blocks movement of Wh words (but not the scope of other NPs) at the substitution nodes of words that delimit extraction islands such as the instances of *that* and the relative pronoun *who* introduced in the previous chapter. These integrity constraints are shown in the updated lexical entries for *that* and the relative pronoun *who* in Figure 6.10.



⟨ Who does John like a book about?
who$(z, a(x, book(x) \wedge about(z)(x), likes(x, john))$ ⟩

Figure 6.11: Derived tree pair and derivation tree for the sentence "Who does John like a book about?" Note that the delay takes place in the syntax as well as the semantics in order for *who* to end up at the front of the sentence.

This still allows us to handle cases of long distance Wh movement out of prepositional phrases, a construction that has posed problems for other systems of TAG semantics [Kallmeyer and Scheffler, 2004]. Consider, for instance, the sentence:

(8) Who does john like a book about?

The analysis given in Kallmeyer and Scheffler [2004] requires a non-local MCTAG, while our analysis is straightforward using the lexical items we have already introduced. The analysis this sentence is given in Figure 6.11.

6.5 Conclusion

The chapter employs an extension to TL-MCTAG, limited delay V-TAG, to handle certain constructions in which the tree-local domain of locality is too constraining to capture the necessary relationship between the syntax and semantics of a sentence. Each of these cases arise when a scope-taking element takes scope outside of the usual tree-local domain of locality. LDV-TAG is well-suited to handle this because it allows analyses that permit non-local adjunction only for the scope parts of certain lexical items. Because LDV-TAG extends TL-MCTAG in a limited way, it also allows the maintenance of the analyses presented in the previous chapter and maintains a similar processing efficiency to that of TL-MCTAG.

Chapter 7

STAG for Machine Translation

7.1 Introduction

Machine translation research has gone through two distinct phases. Initially translation was most successfully performed via the engineering of natural language grammars that mapped the syntax and lexicon of one language into that of another language using detailed knowledge of the structure and vocabularies of both languages. However, the construction of such systems required a very large amount of work by skilled model designers and, even with the large input of resources, resulted in systems that had unacceptably narrow coverage. In the second phase, statistical machine translation (SMT) became ascendant. Using large quantities of data and simple, efficient statistical models, broad coverage translation systems could be developed quickly and without any requirement of skilled work from people versed in the languages to be translated. Although at first the quality of the translations was quite low, more recent advances in the methods of sentence alignment, word

alignment, translation model, and, lately, phrase-based and hierarchical phrase-based translation, have resulted in state-of-the-art systems that generally produce acceptable translations where sufficient training data for the language pair is available.

However, the incremental improvements in SMT are getting smaller over time as the sentences that are mistranslated are narrowed down to those where the syntactic divergence between the two sentences cannot be captured by the model being used by the translation system. The crux of this problem lies in the trade-off between efficiency and syntactic complexity inherent in the statistical machine translation model. Efficient translation depends on a simple model of transfer from a source language to a target language. However, an examination of sentences in translation reveals that a simple correspondence is not always sufficient to capture the full extent of the relationship between two sentences that are translations of each other.

This chapter proposes as as yet unfinished program of research that reaches back to draw on the wisdom of the first phase of machine translation research and combine it with current SMT techniques to create a hybridized, syntax-aware, synchronous grammar-based system that has broad coverage but also exhibits the capability to handle the more difficult sentence pairs that are beyond the scope of what word- and phrase-based translation systems can translate. The system we propose has two main components. First, it has a statistically induced substrate that uses a polynomially parsable constrained version of synchronous Tree Insertion Grammar (STIG) to generate a broad coverage MT grammar. Second, we hybridize the grammar by adding syntactically motivated tree pairs that allow the complete system to capture more complex linguistic correspondences between the two languages in the pair. These

additional tree pairs follow the spirit of the hand-crafted MT phase discussed above, but not the method of generation. Rather than generating these pairs by hand, we extract them automatically from two sources. First, we propose using the as yet untapped resource of bilingual dictionaries, which give phrasal translation examples for the entries that can be automatically converted into tree pair templates. Second, we will extract lexical tree pairs from full parse trees for aligned sentence pairs. To perform this extraction we rely on our own work on minimal factorization of trees as well as that of others on the extraction of trees from treebank data. Once the two levels of the system are present, they can be combined to form one single SMT system. The method of combination of the two levels is an open question for our research. One promising option is to reserve probability for the syntactically motivated trees and then retrain the entire system using both the normal form trees of the statistical substrate and the more complex syntactically motivated trees.

A schematic diagram of the structure of the complete MT system is given in Figure 7.1. In Section 7.2 I present the statistical substrate for the system and some preliminary evaluations of it in some detail. In Section 7.3 I briefly introduce proposals for generating syntactically motivated trees by harvesting from bilingual dictionaries and extracting from treebanks. In Section 7.4, I propose two methods for combining the substrate and the lexicon of statistically motivated trees into a single hybrid system. I conclude in Section 7.5.

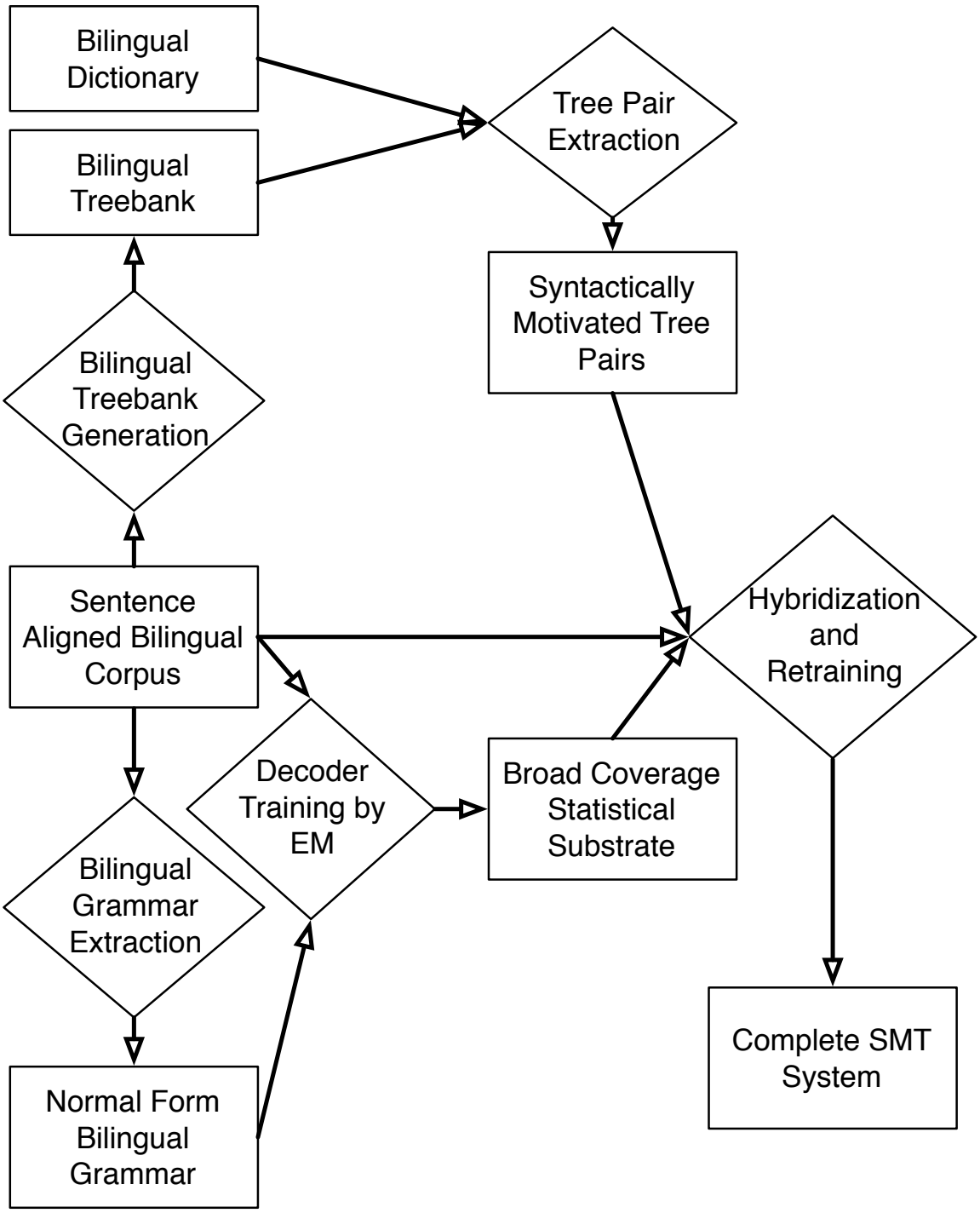


Figure 7.1: A schematic diagram showing the structure of the proposed MT system

7.2 The Statistically Induced Substrate

The statistically induced substrate of the system differs substantially from a typical SMT system because it contemplates hybridization of the system—the addition of syntactically-motivated trees—from the outset. Most SMT systems use the finite-state transducer as a base formalism because finite-state mechanisms can be trained very efficiently on large corpora. However, these systems do not provide any clear way to incorporate additional information about correspondences between words or phrases that may be useful in improving translations. By basing this system on a synchronous grammar in which all translation is done by parsing, we leave open the possibility of adding additional tree pairs to the lexicon that capture more complex lexical and syntactic relationships.

Recent work in statistical machine translation by parsing has identified a set of characteristics an ideal base formalism should have for the translation task [Melamed, 2003, Melamed et al., 2004, Melamed, 2004]. What is desired is a formalism that has the substitution-based hierarchical structure of context-free grammars and the lexical relationship potential of n-gram models. Further, it should allow for discontinuity in phrases and be synchronizable, to allow for multilinguality. Finally, in order to support automated induction, it should allow for a probabilistic variant, and a reasonably efficient parsing algorithm. The more expressive and flexible a formalism is, the less efficient parsing of it will be. Therefore, the primary trade-off to be made is between parsing efficiency on one hand and the rest of the desired characteristics on the other. However, even among formalisms with the same parse complexity, some formalisms better satisfy the desiderata than others.

Finite-state word-based models, such as IBM Model 5 [Brown et al., 1993], use a base formalism that allows for synchronization, probabilistic variants, very efcient processing, and good ability to capture lexical and bilexical relationships. However, they are limited by the inability to use hierarchical information in the interlingual mapping. The ability to incorporate hierarchical information by substitution of subparts is the hallmark of context-free grammars. A natural approach, then is to incorporate synchronization of context-free structures to allow for these kinds of mappings. However, probabilistic context-free grammars (PCFG) are well known to perform poorly as language models compared to nite-state models; they gain the ability to substitute according to abstract categories at the expense of stating lexical relationships directly. Although arbitrary CFGs can be weakly lexicalized by other CFGs, this can require changing the shape of the derived trees produced, and more critically, changes the structure of the derivation [Schabes and Waters, 1993, 1995]. Because synchronization requires substantial isomorphism of the derivation trees, synchronization of lexicalized CFGs becomes problematic. A different form of hybridization, combining both a CFG-based grammar and n-gram-based phrases is the current state-of-the-art [Chiang, 2007].

Tree-adjoining grammars and their synchronous variants (STAG) are natural choices to capture lexically-based dependencies while also allowing the substitution of subparts. Importantly, Schabes and Waters [1995] show that TAG can lexicalize CFG without changing the trees produced. That is, given a CFG a lexicalized TAG can be constructed that will produce the same set of derived structures produced by the CFG. Because each elementary tree contains a lexical item, the operations

of substitution and adjunction implicitly manifest a lexical relationship. In addition, the two operations, substitution and adjunction, are exactly what is needed to handle noncontiguity.

However, as addressed in the early chapters of this thesis, the TAG formalism's additional expressivity leads to additional processing complexity. Because training an MT system based on synchronous TAG would require repeated parsing of the training corpus, this time complexity is prohibitive. Tree-insertion grammars (TIG) are a computationally attractive alternative to TAG [Schabes and Waters, 1993]. TIGs are similar to TAGs except that restrictions are placed on the form of elementary trees and on the adjunction operation that cause it to have CFG-equivalent expressivity and parsing complexity. Schabes and Waters [1995] demonstrate that TIG, like TAG, can lexicalize CFGs without changing the shape of the trees produced. Hwa [2001] shows that a probabilistic variant of TIG can have language modeling performance at the level of bigram models thereby capturing lexical relationships, while also retaining the advantages of CFGs in capturing syntactic structure.

The advantage of STIG as a formalism for MT is its naturalness in describing relations between constructions in different languages. Nonetheless, to make use of that ability it must be embeddable in a system that can show at least the robustness of performance of the nite-state methods that have become standard. Our induced probabilistic STIG is structured as a normal form grammar in which adjunction parameters are estimated by expectation maximization. The normal forms specify both the shape of the trees in the tree pair as well as the links between them. For every observed word cooccurence in the training set, we introduce one of each of the

normal form auxiliary tree pairs anchored by the cooccurring words. The parser learns the adjunction parameters unsupervised on a bilingual corpus using an adaptation of the PCFG inside-outside algorithm developed by Lari and Young [1991].

In this section I first introduce synchronous Tree-Insertion Grammar (Section 7.2.1) and present a parser for a polynomially-parsable restricted subset of it (Section 7.2.2). I then describe the implementation of a probabilistic system based around the parser that uses expectation maximization to learn a grammar from a corpus (Sections 7.2.3 and 7.2.4) and present our preliminary empirical results (Section 7.2.5).

7.2.1 Synchronous Tree-Insertion Grammars

TIGs are similar to TAGs except that restrictions are placed on the form of elementary trees and on the adjunction operation. In particular, the foot node of an auxiliary tree is required to be at the left or right edge of the frontier, so that all textual material dominated by the spine will fall to the right or left, respectively, of the foot. The auxiliary trees can thus be classified as either right or left auxiliary trees, respectively, as determined by the location of the non-foot material.

To maintain the invariant that textual material falls only on a single side of the spine, adjunction must be restricted so that left auxiliary trees may not adjoin into a node on the spine of a right auxiliary tree and vice versa. This prevents the formation of "wrapping" trees in which there are terminal symbols on both sides of the foot node. To implement this restriction we augment adjuction links to indicate a required side of the node where they appear. Substitution links have no side. Notationally, we accomplish this by placing adjunction links on the side of the node on which the

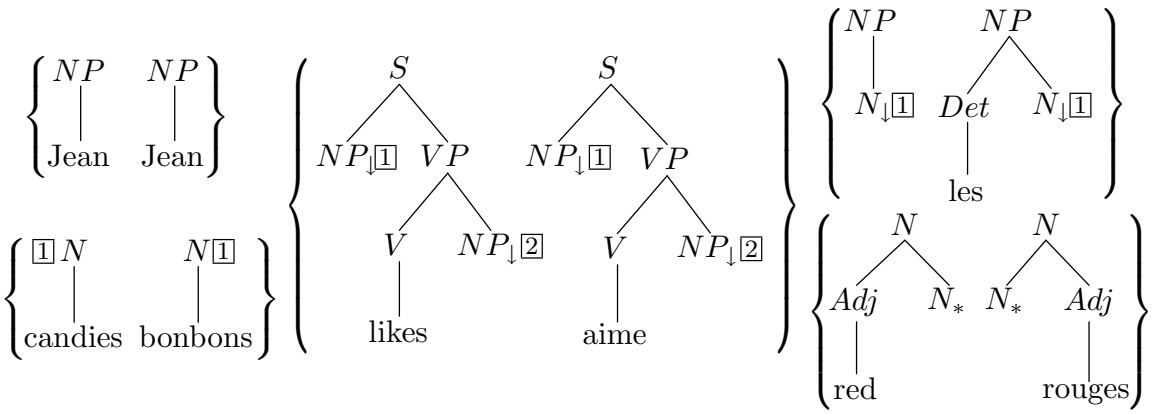


Figure 7.2: An STIG for a sample English/French grammar fragment. Note that although there is a tree with no lexical material, all tree pairs are lexicalized.

adjunction must take place. That is, a link appearing on the left (resp. right) side of a node can only be used by an auxiliary tree with its foot at its right (resp. left) edge. The side of a node on which a substitution link appears carries no special meaning.

This restriction coupled with the requirement that all elementary auxiliary trees be non-wrapping is sufficient to limit the formalism to context-free expressivity and $O(n^3)$ parsability. In addition, Schabes and Waters [1993] demonstrate that TIG, like TAG, can lexicalize context-free grammars without changing the shape of the trees produced. For further background and discussion of TIGs and LTIGs, see the papers by Schabes and Waters [1993, 1995] and Hwa [2001].

Synchronous TIG (STIG) extends TIG just as STAG extends TAG, by making elementary structures pairs of TIG trees with links between particular nodes in those trees. Figure 7.2 contains a sample English/French grammar fragment and Figure 7.3 shows the derivation of the paired sentences: "Jean likes red candies" and "Jean aime les bonbons rouges".

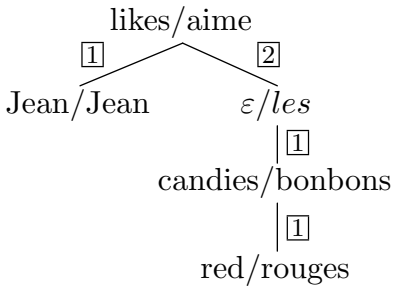


Figure 7.3: The derivation of "Jean really likes candies."/"Jean aime les bonbons rouges."

In support of the hypothesis about the utility of STIG in providing the properties desired in a synchronous grammar formalism for engineering translation systems, Hwa [2001] shows that a probabilistic version of TIG can have language modeling performance at the level of bigram models thereby capturing lexical relationships, while also retaining the advantages of context-free grammars in capturing syntactic structure. A STIG can easily express lexically-based dependencies, can be parsed in $O(n^6)$ time when restricted in ways described below, and can handle both the substitution and adjunction requirements described above. Thus, a probabilistic version of STIG seems to possess all of the properties that we would like as the basis for a the substrate of our syntax-aware translation system.

In the succeeding sections, we develop parsing, parameter estimation, and translation algorithms for probabilistic synchronous TIGs. We follow the useful synchronous parsing framework laid out by Melamed et al. [2004]. Following Melamed, we distinguish multilingual *parsing*, in which a pair of sentences is analyzed as per a multilingual grammar, from *translation*, in which a single sentence is analyzed as per a multilingual grammar so as to determine all multilingual tuples in which the sentence

is admitted. In addition, Melamed et al. [2004] breaks generalized parsers into four distinct pieces, each of which may be analyzed separately: a grammar, a logic, a semiring, and a search strategy. The grammar is the language-specific set of rules and/or lexical items used by the parser. The logic is a set of inference rules that determine how the items of the grammar may be combined to form a derivation. The semiring determines what is computed by the parser as it parses; for instance, it might determine simple acceptability of a sentence by a grammar or it might determine the probability of the sentence being produced by the grammar. The search strategy refers to the order in which the inference rules are applied when more than one rule can apply.[1]

7.2.2 Parsing Synchronous Tree-Insertion Grammars

For reasons to be discussed below, the choice of parsing algorithm has significant consequences. Schabes and Waters [1995] give an Earley-style parsing algorithm for TIGs. Schabes and Waters [1993] give a CKY-style parsing algorithm for a slight variant, lexicalized context-free grammars (LCFG), which can straightforwardly be used to create a CKY-style parsing algorithm for TIGs. We use CKY-style parsing because of the simplicity and clarity of the algorithm and for consistency with parsers presented earlier in this work. Figure 7.4 presents inference rules for a CKY-style parsing algorithm for TIG. This algorithm is an adaptation of the algorithm given in Schabes and Waters [1993] and will appear similar to the TAG parsing algorithm

[1]Melamed draws on previous work by Shieber et al. [1994] for his inference-rule-based separation of logics, grammars, and search strategy. He draws on previous work by Goodman [1999] for the method of parameterizing the parser by a semiring.

presented in Chapter 1. The primary difference between the TAG and TIG parsers is that TIG items do not correspond to subtrees containing gaps, so they require only two indices to demarcate the left and right edges of the input string that they span. To reduce the number of inference rules needed, we make use of an interval union operation $\cup_{\mathtt{x}}$ distinct from the union operation introduced for our earlier parsers. It is parameterized by the order in which the intervals abut, where `x` is either `L` or `R`, defined by

$$\begin{aligned} (i,j) \cup_{\mathtt{L}} (j,k) &= (i,k) \\ (j,k) \cup_{\mathtt{R}} (i,j) &= (i,k) \quad , \end{aligned}$$

and is otherwise undefined.

As discussed in Chapter 4, parsing of any synchronous variant of a CFG-equivalent formalism is NP-complete. Because induction of a probabilistic grammar from data using expectation maximization requires repeated parsing of the corpus, we require a polynomial parsing algorithm with a practically tractable polynomial degree. As a result, we restrict STIG to exclude the configurations of links that can lead to exponential parsing behavior. The critical issue is that links may be intertwined in such a way that the number of discontinuous spans the parser must keep track of in order to ensure adherence to locality restrictions can grow exponentially during parsing. By placing an arbitrary bound on the number of discontinuous spans that may be maintained simultaneously, we reduce the parsing complexity to a polynomial in the length of the input string but also exclude derivations that are valid STIG derivations but violate the bound.

We implement the strictest version of the bound: each item of the parser represents

Item Form:	$\langle \alpha @ a \triangleright \ell, (i,j) \rangle$	
Goal:	$\langle \alpha @ \epsilon \triangleright \_, (0,n) \rangle$	$\mathsf{Init}(\alpha)$ $\mathsf{Label}(\alpha @ \epsilon) = S$
Axioms:		
Terminal Axiom:	$\langle \alpha @ a \triangleright \_, (i,i+1) \rangle$	$\mathsf{Label}(\alpha @ a) = w_{i+1}$
Empty Axiom:	$\langle \alpha @ a \triangleright \_, (i,i) \rangle$	$\mathsf{Label}(\alpha @ a) = \epsilon$
Foot Axiom:	$\langle \alpha @ Ft(\alpha) \triangleright \_, (i,i) \rangle$	$\mathsf{Aux}(\alpha)$
Inference Rules:		
Complete Unary:	$\dfrac{\langle \alpha @ (a \cdot 1) \triangleright \_, (i,j) \rangle}{\langle \alpha @ a \triangleright \ell, (i,j) \rangle}$	$\alpha @ (a \cdot 2)$ undefined $\mathsf{Link}(\alpha @ a) = \ell$
Complete Binary:	$\dfrac{\langle \alpha @ (a \cdot 1) \triangleright \_, I \rangle, \langle \alpha @ (a \cdot 2) \triangleright \_, J \rangle}{\langle \alpha @ a \triangleright \ell, I \cup_{\mathtt{L}} J \rangle}$	$\mathsf{Link}(\alpha @ a) = \ell$
Adjoin:	$\dfrac{\langle \beta @ \epsilon \triangleright \_, I \rangle, \langle \alpha @ a \triangleright \boxed{x}, J \rangle}{\langle \alpha @ a \triangleright \_, I \cup_{\mathtt{x}} J \rangle}$	$\mathsf{Adj}(\alpha @ a \triangleright \boxed{x}, \beta, \mathtt{x})$
No Adjoin:	$\dfrac{\langle \alpha @ a \triangleright \boxed{x}, (i,j) \rangle}{\langle \alpha @ a \triangleright \_, (i,j) \rangle}$	
Substitute:	$\dfrac{\langle \beta @ \epsilon \triangleright \_, (i,j) \rangle}{\langle \alpha @ a \triangleright \_, (i,j) \rangle}$	$\mathsf{Link}(\alpha @ a) = \boxed{x}$ $\mathsf{Sub}(\alpha @ a \triangleright \boxed{x}, \beta)$

Figure 7.4: Inference rules for the CKY algorithm for TIG.

two synchronized subtrees that each dominate a single contiguous span. This limits the polynomial degree of the parser to $O(n^6)$, a reasonably tractable bound for our application.

An interesting aspect of enforcing this limitation is that the order in which the nodes of the trees are traversed affects which configurations of links are excluded by

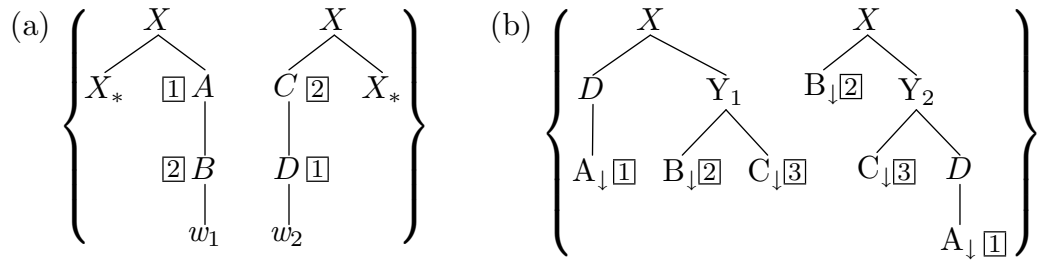


Figure 7.5: STIG links that pose a problem for CKY parsing.

the parser. This is most easily illustrated by an example, such as the tree pair in Figure 7.5(a).

In CKY parsing, traversal of the trees proceeds from all the leaves simultaneously up to the root of the tree. When the parsing algorithm has finished processing all children of a node, it proceeds to the parent. Thus, in order to make use of the link between node B and node C (in Figure 7.5(a)), the parsing algorithm must first finish with node D. However, in order to make use of the link between node A and node D, the parser must first finish with node B. Thus, although a tree pair such as the one in Figure 7.5 is permitted by the STIG formalism, the parser will rule out any derivation that makes use of both of links in the tree pair.

In Earley's algorithm, the nodes are parsed in a top-down, left-first order; the left branches precede the right branches and the left sides of nodes precede the right sides of nodes. Thus, the trees in Figure 7.5(a) would be parsable using a version of Earley's algorithm for STIG, because the right side of node B and the left side of node C would both be reached before the right side of node A and the left side of node D. However, only derivations that make use of a combination of left and right adjunctions that correspond to the traversal order would be generated.

Because of the branching structure of trees, unparsable configurations can arise even with nodes that have no links between them. When two synchronized pairs of nodes cross each other across the boundaries of a branch in the tree, the parser can also become stuck. This problem is illustrated in Figure 7.5(b). In order to perform a sibling concatenation to reach nodes Y_1 and Y_2, the derivation must have reached all of their daughter nodes. If a pair of nodes, such as those labeled D, are not both daughters or descendants of Y_1 and Y_2, the derivation will not be able to proceed. Note that although the nodes labeled D are not explicitly linked, because they dominate nodes that are synchronized with each other, they will be synchronized in a single item when the parser reaches them. Thus the problem persists even when the pairing is not of a daughter of Y_2 with an ancestor of Y_1 but also any descendant of Y_2 with an ancestor of Y_1.

To formalize the problematic situation, we introduce the notion of a cover. We say that set of nodes S covers an ancestor node X if all descendants of X that are paired nodes are contained in S. We prohibit tree structures in which the set of nodes that cover a node, here Y_1, are not paired with the set of nodes that cover Y_1's paired node Y_2.

We define a restricted subset of STIG: **restricted STIG** (RSTIG), for which we assert that the given CKY parsing algorithm is complete. An RSTIG is a STIG where the linking relation in the elementary trees obeys the following rules:

- If a particular elementary tree pair contains a link between node A and node B, it may not also contain a link between: (a) an ancestor of A and a descendant of B, or (b) an ancestor of B and a descendant of A.

Item Form:

$$\{\mathtt{s} : \langle \alpha_{\mathtt{s}} @ a_{\mathtt{s}} \triangleright \ell_{\mathtt{s}}, (i_{\mathtt{s}}, j_{\mathtt{s}}) \rangle, \mathtt{t} : \langle \alpha_{\mathtt{t}} @ a_{\mathtt{t}} \triangleright \ell_{\mathtt{t}}, (i_{\mathtt{t}}, j_{\mathtt{t}}) \rangle\}$$

Goal:

$$\{\mathtt{s} : \langle \alpha_{\mathtt{s}} @ \epsilon \triangleright \_, (0, n_{\mathtt{s}}) \rangle, \mathtt{t} : \langle \alpha_{\mathtt{t}} @ \epsilon \triangleright \_, (0, n_{\mathtt{t}}) \rangle\} \qquad \mathsf{Init}(\alpha)$$

$$\mathsf{Label}(\alpha_{\mathtt{s}} @ \epsilon) = S_{\mathtt{s}}$$

Axioms: $\mathsf{Label}(\alpha_{\mathtt{t}} @ \epsilon) = S_{\mathtt{t}}$

Terminal Axiom:

$$\{\mathtt{x} : \langle \alpha @ a \triangleright \_, (i, i+1) \rangle, \overline{\mathtt{x}} : \_\} \qquad \mathsf{Label}(\alpha @ a) = w_{i+1}$$

Empty Axiom:

$$\{\mathtt{x} : \langle \alpha @ a \triangleright \_, (i, i) \rangle, \overline{\mathtt{x}} : \_\} \qquad \mathsf{Label}(\alpha @ a) = \epsilon$$

Foot Axiom:

$$\{\mathtt{x} : \langle \alpha @ Ft(\alpha) \triangleright \_, (i, i) \rangle, \overline{\mathtt{x}} : \_\} \qquad \mathsf{Aux}(\alpha)$$

Figure 7.6: The item form, goal item, and axioms for the CKY algorithm for RSTIG

- If a set of nodes form a cover of an ancestor, X, then the nodes with which those nodes are paired must form a cover of any nodes with which X may be paired.

The following definitions apply:

- **Paired.** Two nodes are paired if they may form an item in the course of a derivation.
- **Cover.** A set of nodes S covers an ancestor node X if all descendants of X that are paired nodes are contained in S.

Given this definition, we can define a parser for RSTIG. The inference rules for the CKY-style RSTIG parsing algorithm are shown in Figures 7.6 and 7.7.

Each item is represented as a two element set where the elements have the same form as the items of the TIG parser presented above. For notational convenience, the two members are labeled as source (`s`) or target (`t`). When a rule applies to

Inference Rules:

Complete Unary:

$$\frac{\{\mathtt{x} : \langle \alpha@(a \cdot 1) \triangleright \_, (i,j)\rangle, \overline{\mathtt{x}} : \mathtt{X}\}}{\{\mathtt{x} : \langle \alpha@a \triangleright \ell, (i,j)\rangle, \overline{\mathtt{x}} : \mathtt{X}\}} \quad \begin{array}{l} \alpha@(a \cdot 2) \text{ undefined} \\ \mathsf{Link}(\alpha@a) = \ell \end{array}$$

Complete Binary 1:

$$\frac{\begin{array}{c}\{\mathtt{x} : \langle \alpha_{\mathtt{x}}@(a_{\mathtt{x}} \cdot 1) \triangleright \_, I_{\mathtt{x}}\rangle, \overline{\mathtt{x}} : \langle \alpha_{\overline{\mathtt{x}}}@(a_{\overline{\mathtt{x}}} \cdot 1) \triangleright \_, I_{\overline{\mathtt{x}}}\rangle\}, \\ \{\mathtt{x} : \langle \alpha_{\mathtt{x}}@(a_{\mathtt{x}} \cdot 2) \triangleright \_, J_{\mathtt{x}}\rangle, \overline{\mathtt{x}} : \langle \alpha_{\overline{\mathtt{x}}}@(a_{\overline{\mathtt{x}}} \cdot 2) \triangleright \_, J_{\overline{\mathtt{x}}}\rangle\}\end{array}}{\{\mathtt{x} : \langle \alpha_{\mathtt{x}}@a_{\mathtt{x}} \triangleright \ell_{\mathtt{x}}, I_{\mathtt{x}} \cup_{\mathtt{L}} J_{\mathtt{x}}\rangle, \overline{\mathtt{x}} : \langle \alpha_{\overline{\mathtt{x}}}@a_{\overline{\mathtt{x}}} \triangleright \ell_{\overline{\mathtt{x}}}, I_{\overline{\mathtt{x}}} \cup_{\mathtt{L}} J_{\overline{\mathtt{x}}}\rangle\}} \quad \begin{array}{l} \mathsf{Link}(\alpha_{\mathtt{x}}@a_{\mathtt{x}}) = \ell_{\mathtt{x}} \\ \mathsf{Link}(\alpha_{\overline{\mathtt{x}}}@a_{\overline{\mathtt{x}}}) = \ell_{\overline{\mathtt{x}}} \end{array}$$

Complete Binary 2:

$$\frac{\begin{array}{c}\{\mathtt{x} : \langle \alpha_{\mathtt{x}}@(a_{\mathtt{x}} \cdot 1) \triangleright \_, I_{\mathtt{x}}\rangle, \overline{\mathtt{x}} : \langle \alpha_{\overline{\mathtt{x}}}@(a_{\overline{\mathtt{x}}} \cdot 2) \triangleright \_, I_{\overline{\mathtt{x}}}\rangle\}, \\ \{\mathtt{x} : \langle \alpha_{\mathtt{x}}@(a_{\mathtt{x}} \cdot 2) \triangleright \_, J_{\mathtt{x}}\rangle, \overline{\mathtt{x}} : \langle \alpha_{\overline{\mathtt{x}}}@(a_{\overline{\mathtt{x}}} \cdot 1) \triangleright \_, J_{\overline{\mathtt{x}}}\rangle\}\end{array}}{\{\mathtt{x} : \langle \alpha_{\mathtt{x}}@a_{\mathtt{x}} \triangleright \ell_{\mathtt{x}}, I_{\mathtt{x}} \cup_{\mathtt{L}} J_{\mathtt{x}}\rangle, \overline{\mathtt{x}} : \langle \alpha_{\overline{\mathtt{x}}}@a_{\overline{\mathtt{x}}} \triangleright \ell_{\overline{\mathtt{x}}}, I_{\overline{\mathtt{x}}} \cup_{\mathtt{R}} J_{\overline{\mathtt{x}}}\rangle\}} \quad \begin{array}{l} \mathsf{Link}(\alpha_{\mathtt{x}}@a_{\mathtt{x}}) = \ell_{\mathtt{x}} \\ \mathsf{Link}(\alpha_{\overline{\mathtt{x}}}@a_{\overline{\mathtt{x}}}) = \ell_{\overline{\mathtt{x}}} \end{array}$$

Complete Binary 3:

$$\frac{\begin{array}{c}\{\mathtt{x} : \langle \alpha_{\mathtt{x}}@(a_{\mathtt{x}} \cdot 1) \triangleright \_, I_{\mathtt{x}}\rangle, \overline{\mathtt{x}} : \mathtt{X}_1\}, \\ \{\mathtt{x} : \langle \alpha_{\mathtt{x}}@(a_{\mathtt{x}} \cdot 2) \triangleright \_, J_{\mathtt{x}}\rangle, \overline{\mathtt{x}} : \mathtt{X}_2\}\end{array}}{\{\mathtt{x} : \langle \alpha_{\mathtt{x}}@a_{\mathtt{x}} \triangleright \ell_{\mathtt{x}}, I_{\mathtt{x}} \cup_{\mathtt{L}} J_{\mathtt{x}}\rangle, \overline{\mathtt{x}} : \mathtt{X}_1 \cup \mathtt{X}_2\}} \quad \begin{array}{l} \mathsf{Link}(\alpha_{\mathtt{x}}@a_{\mathtt{x}}) = \ell_{\mathtt{x}} \\ \mathsf{Link}(\alpha_{\overline{\mathtt{x}}}@a_{\overline{\mathtt{x}}}) = \ell_{\overline{\mathtt{x}}} \end{array}$$

Adjoin:

$$\frac{\begin{array}{c}\{\mathtt{s} : \langle \alpha_{\mathtt{s}}@a_{\mathtt{s}} \triangleright \boxed{x}, I_{\mathtt{s}}\rangle, \mathtt{t} : \langle \alpha_{\mathtt{t}}@a_{\mathtt{t}} \triangleright \boxed{x}, I_{\mathtt{t}}\rangle\} \\ \{\mathtt{s} : \langle \beta_{\mathtt{s}}@\epsilon \triangleright \_, J_{\mathtt{s}}\rangle, \mathtt{t} : \langle \beta_{\mathtt{t}}@\epsilon \triangleright \_, J_{\mathtt{t}}\rangle\}\end{array}}{\{\mathtt{s} : \langle \alpha_{\mathtt{s}}@a_{\mathtt{s}} \triangleright \_, I_{\mathtt{s}} \cup_{\mathtt{x}_1} J_{\mathtt{s}}\rangle, \mathtt{t} : \langle \alpha_{\mathtt{t}}@a_{\mathtt{t}} \triangleright \_, I_{\mathtt{t}} \cup_{\mathtt{x}_2} J_{\mathtt{t}}\rangle\}} \quad \begin{array}{l} \mathsf{Adj}(\alpha_{\mathtt{s}}@a_{\mathtt{s}} \triangleright \boxed{x}, \beta_{\mathtt{s}}, \mathtt{x}_1) \\ \mathsf{Adj}(\alpha_{\mathtt{t}}@a_{\mathtt{t}} \triangleright \boxed{x}, \beta_{\mathtt{t}}, \mathtt{x}_2) \end{array}$$

No Adjoin:

$$\frac{\{\mathtt{x} : \langle \alpha@a \triangleright \boxed{x}, (i,j)\rangle, \overline{\mathtt{x}} : \mathtt{X}\}}{\{\mathtt{x} : \langle \alpha@a \triangleright \_, (i,j)\rangle, \overline{\mathtt{x}} : \mathtt{X}\}}$$

Substitute:

$$\frac{\{\mathtt{s} : \langle \beta_{\mathtt{s}}@\epsilon \triangleright \_, (i_{\mathtt{s}}, j_{\mathtt{s}})\rangle, \mathtt{t} : \langle \beta_{\mathtt{t}}@\epsilon \triangleright \_, (i_{\mathtt{t}}, j_{\mathtt{t}})\rangle\}}{\{\mathtt{s} : \langle \alpha_{\mathtt{s}}@a_{\mathtt{s}} \triangleright \_, (i_{\mathtt{s}}, j_{\mathtt{s}})\rangle, \mathtt{t} : \langle \alpha_{\mathtt{t}}@a_{\mathtt{t}} \triangleright \_(i_{\mathtt{t}}, j_{\mathtt{t}})\rangle\}} \quad \begin{array}{l} \mathsf{Link}(\alpha_{\mathtt{s}}@a_{\mathtt{s}}) = \boxed{x} \\ \mathsf{Link}(\alpha_{\mathtt{t}}@a_{\mathtt{t}}) = \boxed{x} \\ \mathsf{Sub}(\alpha_{\mathtt{s}}@a_{\mathtt{s}} \triangleright \boxed{x}, \beta_{\mathtt{s}}) \\ \mathsf{Sub}(\alpha_{\mathtt{t}}@a_{\mathtt{t}} \triangleright \boxed{x}, \beta_{\mathtt{t}}) \end{array}$$

Merge:

$$\frac{\begin{array}{c}\{\mathtt{s} : \langle \alpha_{\mathtt{s}}@a_{\mathtt{s}} \triangleright \ell_{\mathtt{s}}, (i_{\mathtt{s}}, j_{\mathtt{s}})\rangle, \mathtt{t} : \_\} \\ \{\mathtt{s} : \_, \mathtt{t} : \langle \alpha_{\mathtt{t}}@a_{\mathtt{t}} \triangleright \ell_{\mathtt{t}}, (i_{\mathtt{t}}, j_{\mathtt{t}})\rangle\}\end{array}}{\{\mathtt{s} : \langle \alpha_{\mathtt{s}}@a_{\mathtt{s}} \triangleright \ell_{\mathtt{s}}, (i_{\mathtt{s}}, j_{\mathtt{s}})\rangle, \mathtt{t} : \langle \alpha_{\mathtt{t}}@a_{\mathtt{t}} \triangleright \ell_{\mathtt{t}}, (i_{\mathtt{t}}, j_{\mathtt{t}})\rangle\}}$$

Figure 7.7: The inference rules for the CKY algorithm for RSTIG.

only one member of the set without regard for which side, the label `x` is used to represent either `s` or `t` and the label $\overline{\texttt{x}}$ is used to represent the other element. The axioms introduce what may be thought of as unsynchronized items in which only one of the two elements of the set is instantiated. Those rules that do not require synchronization, such as Complete Unary and Complete Binary 3, operate on just one side of the synchronous pair leaving the other side of the pair unchanged.[2] This allows for arbitrary unpaired structure in the elementary trees. Operations such as Adjoin, Substitute, and Complete Binary 1 and 2, require a synchronous pair of nodes in order to properly enforce synchronization. When the parser reaches a link or a location that cannot be processed without synchronization, the Merge rule is applied to form a synchronized item from two unsynchronized items.[3]

We include a No Adjoin rule but note that it would be possible to omit this rule. When the rule is omitted and the parser needs to skip a particular link, it is simulated by adjoining in a special auxiliary tree pair in which each tree is a single node that is both root and foot. Thus, it does not change the shape of the derived tree.[4] This change eliminates the tricky problem of estimating no-adjunction parameters. However, we note that grammar size may increase as a result of this change because

[2]Note that in Complete Binary 3 we make use of the union operator introduced in earlier parsers. This operator takes two inputs and selects whichever one is non-null. If both are null, the result is null. If both are non-null, then the operation fails.

[3]Although as written the Merge rule can apply at any time, we assume a constraint on its application that allows it to be used only when no other rule can be applied. This prevents spurious ambiguity arising from its application.

[4]The addition of empty trees breaks lexicalization of an otherwise lexicalized grammar, but neither removes the linguistic advantages of lexicalization nor the parsing advantage that comes from not allowing adjunctions that don't increase the span of the item. The reason the latter is not a problem is that an empty tree can only adjoin to a link once because the link is then removed. Thus no spurious adjunctions are introduced. In our implementation we do not actually add the empty tree nodes to the chart but instead just make them available in every chart cell with no cover.

the parser can no longer skip any links in a derivation. With the No Adjoin rule in place, trees with unparsable link configurations may exist in the grammar as long as only a parsable subset is used in the derivation. Without it, the lexicon itself may not include any unparsable link configurations.

Allowing Translations of Differing Lengths

The rules presented above allow the introduction of trees with empty anchors, breaking lexicalization. Allowing empty anchors permits us to model cases in which a word in one language translates into the empty string in another language.[5] Even if we restrict the Merge rule to prevent the synchronization of tree pairs that contain no lexical material on either side, this can as much as triple the size of the grammar, which will have a detrimental effect on the space and time complexity of the parsing algorithm as well as the space and time complexity of the expectation maximization algorithm based on it. However, because each operation that combines trees (excepting the adjunction of empty trees discussed above) still increases the cover of the item in at least one of the sentences, the time complexity of the algorithm in terms of sentence length remains the same.

It also affects the time and space complexity of the translation process. The translator has only the source sentence as input and it must guess if there are any tree pairs in the parse that have a tree anchored with the empty string on the source side. Theoretically there can be arbitrarily many such tree pairs in the translation.

[5]It also gives us a simple way to handle cases in which multiword phrases translate to phrases of differing lengths, although this is more elegantly handled by the use of the ability to parse unsynchronized structure. In our implementation we use canonical trees that do not exhibit any non-isomorphic structure in the tree pairs, so in effect we do not make use of the asynchronous rules. They could, however, be useful when the grammar is hybridized with linguistically-motivated trees.

We use a heuristic that limits the size of the target sentence relative to the source sentence to prevent the parser from searching a potentially very large set of possible translations.

7.2.3 Parsing Probabilistic RSTIG

In order to use the RSTIG parser to induce an RSTIG grammar from data, we need to add probabilities to each item, corresponding to the inside probability of the source and target nodes in that item covering their respective parts of the source and target input sentences. Following the Melamed et al. [2004] framework and the work of Goodman [1999] on parameterizing parsers with semirings, we are able to do this quite easily. In this section, we review the concept of a semiring and demonstrate how several useful semirings can be applied to the RSTIG inference rules and grammar.

Semirings

Goodman [1999] demonstrates that the quantities most commonly computed by parsers can all be computed using the same parsing algorithm by simply swapping in a semiring that aids in calculating the particular quantity desired. The basic idea is that the parsing algorithm can be written to compute a quantity, such as acceptance or inside probability, by using a set of semiring operators in certain places. To change the quantity computed is to change the semiring being used.

A semiring contains two operators, a product ($\otimes$) and sum ($\oplus$), and two identities, 0 and 1. We require that $\oplus$ be associative and commutative and that $\otimes$ be associative and distribute over $\oplus$. 0 is an additive identity element and 1 is a mul-

tiplicative identity element. We will write $\langle S, \oplus, \otimes, 0, 1 \rangle$ for the semiring over set S with multiplicative operator $\otimes$ and identity 1 and additive operator $\oplus$ and identity 0.

In addition to the above, Goodman [1999] also includes a requirement that the semiring be complete. A complete semiring also allows for infinite sums that are associative and commutative and allows the multiplicative operator to distribute over infinite sums. Completeness is necessary to handle parsers in which there may be an infinite number of derivations for a particular item. This occurs when an item can derive itself, either directly or through a series of steps. Although all the semirings we describe here are complete, because operations can only take place at links and links are removed after an operation occurs, there will never be an infinite number of derivations of an item in our system.

To parameterize the parser by a semiring, each item and inference rule will have a semiring value associated with it. The semiring value associated with the consequent will be the product of the antecedent items' semiring values and the semiring value of the inference rule itself. When an item is added to the chart, if it is not present in the chart it will be added with this computed value. If it is already present in the chart, the new value will be its semiring value in the chart plus the value computed by the inference rule that just generated it. Note that if we use the Boolean semiring and give each inference rule the semiring value 1, we yield the same results as we did with the unparameterized parsing algorithm.

In addition to replicating the recognizer previously presented, we can also use semirings to compute inside probabilities and Viterbi derivations for the items in the

chart. The inside probability of an item is the probability of that item being produced by the grammar starting at any point within the grammar. This probability is used in expectation maximization to induce a synchronous TIG grammar from data. The Inside semiring is defined as $\langle \mathbb{R}_0^\infty, +, \times, 0, 1 \rangle$. We set the semiring values associated with each inference rule as follows:

- The semiring value associated with all axiom rules will be 1. Since no decisions have been made in the parse at this point, each axiom item has probability 1 of being generated in the position in which it is inserted into the chart.

- The semiring values associated with the Complete Unary, Complete Binary and Merge rules will be 1. They are entirely determined by the structure of the trees themselves. There is nothing probabilistic in their application.[6]

- The semiring value associated with substitution of item B into item A will be the probability that B substitutes into A. Thus there will be a probability distribution over all items that can substitute into A. This probability will be given as part of the grammar and could be learned from a corpus.

- The semiring value associated with adjunction of item B into item A will be the probability that B adjoins into A. Again, there will be a probability distribution over all items that can adjoin into A given as part of the grammar or learned from a corpus. Note that we do not need to maintain a separate probability of no

[6]As noted above, a requirement on the Merge rule is necessary to constrain its application so that it does not give rise to spurious ambiguity. In our implementation, due to the highly constrained structure of our canonical trees we actually use a simplified version of the parsing rules in which every node is synchronized. This may be thought of as equivalent to applying the merge rule immediately following the introduction of two unsynchronized items by the axioms.

Semiring	S	$\oplus$	$\otimes$	0	1
Recognition	{FALSE, TRUE}	$\vee$	$\wedge$	FALSE	TRUE
Inside	$[0,1]$	$+$	$\times$	0	1
Viterbi	$[0,1]$	max	$\times$	0	1
Viterbi Derivation	$[0,1] \times \mathcal{D}$	$\max_{vit}$	$(\times, \cdot)$	$(0, \emptyset)$	$(1, \langle\rangle)$

Figure 7.8: Summary of the semirings used. $\mathcal{D}$ is the set of derivations.

adjunction occurring because that probability is maintained as the probability of the appropriate empty tree within this distribution.

The Viterbi derivation is the derivation tree of the most probable parse for a given input. This value will be of use when we use our induced grammar to translate new sentences. The Viterbi derivation semiring is given in Figure 7.8 along with the others used in the translation system. The elements of the semiring are pairs consisting of the probability of the item and the derivation of the item. The additive operator takes the maximum probability item and its associated derivation. Although retaining only a single derivation when multiple derivations have the same probability destroys associativity, Goodman [1999] notes that in practice this is not a problem.[7] The multiplicative operator is just $\times$ for the probability and concatenation for the derivations. The additive identity is 0 for the probability and the empty set for the derivation. The multiplicative identity is 1 for the probability and the empty derivation. Concatenation with the empty derivation is an identity operation. The semiring values associated with the rules will use the same probabilities as with the Inside semiring. The pairs will be filled out with the empty derivation so that they

[7]To maintain associativity, we just keep a forest of best derivations rather than a single best derivation.

will be identity elements when multiplied in.

RSTIG Parsing Parameterized by a Semiring

The semiring values associated with the application of each inference rule are the parameters that must be learned from the corpus. Thus we present these parameters as well. Figure 7.9 contains a sampling of the inference rules modified to include the calculation of the semiring values associated with each item.

The semiring value calculated for the string is then the sum of the values associated with the goal items multiplied by their probability of rooting the corresponding derivation:

$$\sum_{\{\mathtt{s}:\langle\alpha_\mathtt{s}@\epsilon\triangleright_-,(0,n_\mathtt{s})\rangle,\mathtt{t}:\langle\alpha_\mathtt{t}@\epsilon\triangleright_-,(0,n_\mathtt{t})\rangle,p\}} SP(\alpha)\otimes p \quad .$$

To make this more concrete, consider the application of the Inside semiring to the adjunction operation. The inside probability of the consequent item is the probabilities of the individual antecedent items, $p_1 \otimes p_2$, multiplied by the probability, p_a, of those two items being adjoined together.

The semiring values associated with each rule are the parameters that will be estimated by the expectation maximization algorithm as described in Section 7.2.4. Since the application of the semiring makes explicit the parameters that will have to be estimated, these parameters are explained here. There are three categories of parameters to be estimated:

(i) **Adjunction Parameters.** Each adjunction link has an associated probability distribution over all of the tree pairs that could adjoin at that link. We further restrict the use of links by having each link specify on which side of the node

Item Form:

$\{\mathtt{s} : \langle \alpha_{\mathtt{s}} @ a_{\mathtt{s}} \triangleright \ell_{\mathtt{s}}, (i_{\mathtt{s}}, j_{\mathtt{s}}) \rangle, \mathtt{t} : \langle \alpha_{\mathtt{t}} @ a_{\mathtt{t}} \triangleright \ell_{\mathtt{t}}, (i_{\mathtt{t}}, j_{\mathtt{t}}) \rangle, p\}$

Goal:

$\{\mathtt{s} : \langle \alpha_{\mathtt{s}} @ \epsilon \triangleright \_, (0, n_{\mathtt{s}}) \rangle, \mathtt{t} : \langle \alpha_{\mathtt{t}} @ \epsilon \triangleright \_, (0, n_{\mathtt{t}}) \rangle, p\}$ $\quad \mathsf{Init}(\alpha)$, $\mathsf{Label}(\alpha_{\mathtt{s}} @ \epsilon) = S_{\mathtt{s}}$, $\mathsf{Label}(\alpha_{\mathtt{t}} @ \epsilon) = S_{\mathtt{t}}$

Axioms:

Terminal Axiom:

$\{\mathtt{x} : \langle \alpha @ a \triangleright \_, (i, i+1) \rangle, \overline{\mathtt{x}} : \_, 1\}$ $\quad \mathsf{Label}(\alpha @ a) = w_{i+1}$

Inference Rules:

Complete Binary 1:

$$\frac{\{\mathtt{x} : \langle \alpha_{\mathtt{x}} @ (a_{\mathtt{x}} \cdot 1) \triangleright \_, I_{\mathtt{x}} \rangle, \overline{\mathtt{x}} : \langle \alpha_{\overline{\mathtt{x}}} @ (a_{\overline{\mathtt{x}}} \cdot 1) \triangleright \_, I_{\overline{\mathtt{x}}} \rangle, p_1\}, \quad \{\mathtt{x} : \langle \alpha_{\mathtt{x}} @ (a_{\mathtt{x}} \cdot 2) \triangleright \_, J_{\mathtt{x}} \rangle, \overline{\mathtt{x}} : \langle \alpha_{\overline{\mathtt{x}}} @ (a_{\overline{\mathtt{x}}} \cdot 2) \triangleright \_, J_{\overline{\mathtt{x}}} \rangle, p_2\}}{\{\mathtt{x} : \langle \alpha_{\mathtt{x}} @ a_{\mathtt{x}} \triangleright \ell_{\mathtt{x}}, I_{\mathtt{x}} \cup_{\mathtt{L}} J_{\mathtt{x}} \rangle, \; \overline{\mathtt{x}} : \langle \alpha_{\overline{\mathtt{x}}} @ a_{\overline{\mathtt{x}}} \triangleright \ell_{\overline{\mathtt{x}}}, I_{\overline{\mathtt{x}}} \cup_{\mathtt{L}} J_{\overline{\mathtt{x}}} \rangle, p_1 \otimes p_2\}}$$

$\mathsf{Link}(\alpha_{\mathtt{x}} @ a_{\mathtt{x}}) = \ell_{\mathtt{x}}$, $\mathsf{Link}(\alpha_{\overline{\mathtt{x}}} @ a_{\overline{\mathtt{x}}}) = \ell_{\overline{\mathtt{x}}}$

Adjoin:

$$\frac{\{\mathtt{s} : \langle \alpha_{\mathtt{s}} @ a_{\mathtt{s}} \triangleright \boxed{x}, I_{\mathtt{s}} \rangle, \mathtt{t} : \langle \alpha_{\mathtt{t}} @ a_{\mathtt{t}} \triangleright \boxed{x}, I_{\mathtt{t}} \rangle, p_1\} \quad \{\mathtt{s} : \langle \beta_{\mathtt{s}} @ \epsilon \triangleright \_, J_{\mathtt{s}} \rangle, \mathtt{t} : \langle \beta_{\mathtt{t}} @ \epsilon \triangleright \_, J_{\mathtt{t}} \rangle, p_2\}}{\{\mathtt{s} : \langle \alpha_{\mathtt{s}} @ a_{\mathtt{s}} \triangleright \_, I_{\mathtt{s}} \cup_{\mathtt{x}_1} J_{\mathtt{s}} \rangle, \; \mathtt{t} : \langle \alpha_{\mathtt{t}} @ a_{\mathtt{t}} \triangleright \_, I_{\mathtt{t}} \cup_{\mathtt{x}_2} J_{\mathtt{t}} \rangle, p_1 \otimes p_2 \otimes p_a\}}$$

$\mathsf{Adj}(\alpha_{\mathtt{s}} @ a_{\mathtt{s}} \triangleright \boxed{x}, \beta_{\mathtt{s}}, \mathtt{x}_1, p_a)$, $\mathsf{Adj}(\alpha_{\mathtt{t}} @ a_{\mathtt{t}} \triangleright \boxed{x}, \beta_{\mathtt{t}}, \mathtt{x}_2, p_a)$

Substitute:

$$\frac{\{\mathtt{s} : \langle \beta_{\mathtt{s}} @ \epsilon \triangleright \_, (i_{\mathtt{s}}, j_{\mathtt{s}}) \rangle, \mathtt{t} : \langle \beta_{\mathtt{t}} @ \epsilon \triangleright \_, (i_{\mathtt{t}}, j_{\mathtt{t}}) \rangle, p\}}{\{\mathtt{s} : \langle \alpha_{\mathtt{s}} @ a_{\mathtt{s}} \triangleright \_, (i_{\mathtt{s}}, j_{\mathtt{s}}) \rangle, \mathtt{t} : \langle \alpha_{\mathtt{t}} @ a_{\mathtt{t}} \triangleright \_(i_{\mathtt{t}}, j_{\mathtt{t}}) \rangle, p \otimes p_s\}}$$

$\mathsf{Link}(\alpha_{\mathtt{s}} @ a_{\mathtt{s}}) = \boxed{x}$, $\mathsf{Link}(\alpha_{\mathtt{t}} @ a_{\mathtt{t}}) = \boxed{x}$, $\mathsf{Sub}(\alpha_{\mathtt{s}} @ a_{\mathtt{s}} \triangleright \boxed{x}, \beta_{\mathtt{s}}, p_s)$, $\mathsf{Sub}(\alpha_{\mathtt{t}} @ a_{\mathtt{t}} \triangleright \boxed{x}, \beta_{\mathtt{t}}, p_s)$

Figure 7.9: Representative inference rules for the CKY algorithm for PRSTIG

the adjunction must take place. This means that only a subset of the total tree pairs will be represented in any adjunction link's distribution. In addition, the nonterminal symbols at the nodes linked further restrict which tree pairs may adjoin.

(ii) **Substitution Parameters.** Each substitution link has an associated proba-

bility distribution over all of the tree pairs that could substitute at that link. The nonterminal symbols at the linked nodes restrict the tree pairs that will be represented in this distribution.

(iii) **Start Tree Parameter.** This parameter specifies for each initial tree pair the of its serving as the root of a derivation, and is associated with the root nodes of the paired trees with the initial tree pairs of the grammar that are rooted in the start symbols of the grammars. This value is represented in the rules by a value for each pair of paired root nodes, $SP(\alpha)$, that is multiplied into the semiring value of the goal item.

7.2.4 Induction from Sentence-Aligned Text

The synchronous parser presented above comes to life when put in the context of an algorithm that can estimate the necessary parameters of the grammar. In this section we review the Inside-Outside algorithm, which is a special case of the expectation maximization algorithm used for estimating the parameters of a grammar [Prescher, 2001]. We apply the algorithm to synchronous tree-insertion grammars in a straightforward generalization of the algorithm presented by Hwa [2001].

The Inside-Outside Algorithm

The Inside-Outside algorithm was proposed by Baker [1979] and refined by Lari and Young [1991] as a method for performing maximum likelihood estimation for probabilistic context-free grammars. The objective of the algorithm is to estimate a probability to associate with each rule of a grammar so that the grammar can then

Initialize the parameters of the grammar randomly
Repeat until convergence:
    E Step:
        Compute the Inside and Outside probabilities for the grammar
    M Step:
        Update the parameters to maximize the likelihood of the training data

Figure 7.10: High-Level View of the Inside-Outside Algorithm.

be used to produce the most likely parse of unseen sentences.

In an unambiguous grammar these probabilities can be estimated directly by parsing a training corpus and keeping track of the number of times each rule is used. These counts can then be normalized to produce a probability to associate with each rule of the grammar using the following formula [Jurafsky and Martin, 2000]:

$$P(\alpha \rightarrow \beta \mid \alpha) = \frac{\text{Count}_{(\alpha \rightarrow \beta)}}{\sum_{\gamma} \text{Count}_{(\alpha \rightarrow \gamma)}} = \frac{\text{Count}_{(\alpha \rightarrow \beta)}}{\text{Count}_{(\alpha)}}$$

However, grammars are rarely unambiguous and our initial grammar will certainly be ambiguous. Thus, these values cannot be computed directly. Instead we must keep track of all of the different parses for a given sentence and a weight for each of those parses. The Inside-Outside algorithm allows us to do just this. It begins with randomly initialized probabilities for each parameter (Initialization Step). It then uses these rules to parse a training corpus and calculate the expected frequency with which each rule is used (E Step). These expected frequencies are then used to compute new probability estimates for the rules (M Step). The process continues until the rule probabilities converge to a local maximum for the training corpus. A high-level description of the algorithm is given in Figure 7.10.

In the E step we calculate the inside and outside probabilities for each of the parameters of the grammar. In the case of PCFGs, the inside probabilities correspond

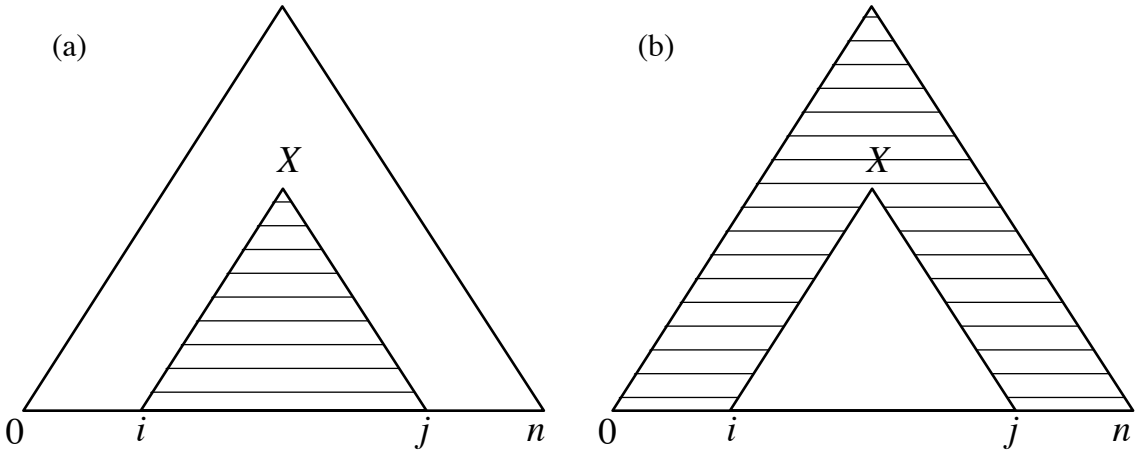


Figure 7.11: The (a) inside and (b) outside probability of the nonterminal symbol X deriving the substring from i to j.

to the probability that a given nonterminal in the grammar derives a given substring of the current input sentence. The outside probabilities correspond to the probability that the entire input sentence is derived with a given nonterminal in the grammar covering a given substring in the input sentence. These two quantities are depicted as the striped areas in Figure 7.11.

In the M step we use the inside and outside probabilities calculated in the E Step to update the parameter probabilities. When multiplied together, the inside and outside probabilities for a particular rule and sentence give the probability that the rule was used in the parse of the sentence. When divided by the probability of the sentence given all possible parses, we get the updated probability for the rule.

The Inside-Outside Algorithm for STIG

The biggest decision to be made in adapting the Inside-Outside algorithm to STIG is to determine the tree pairs that make up the grammar and over which the

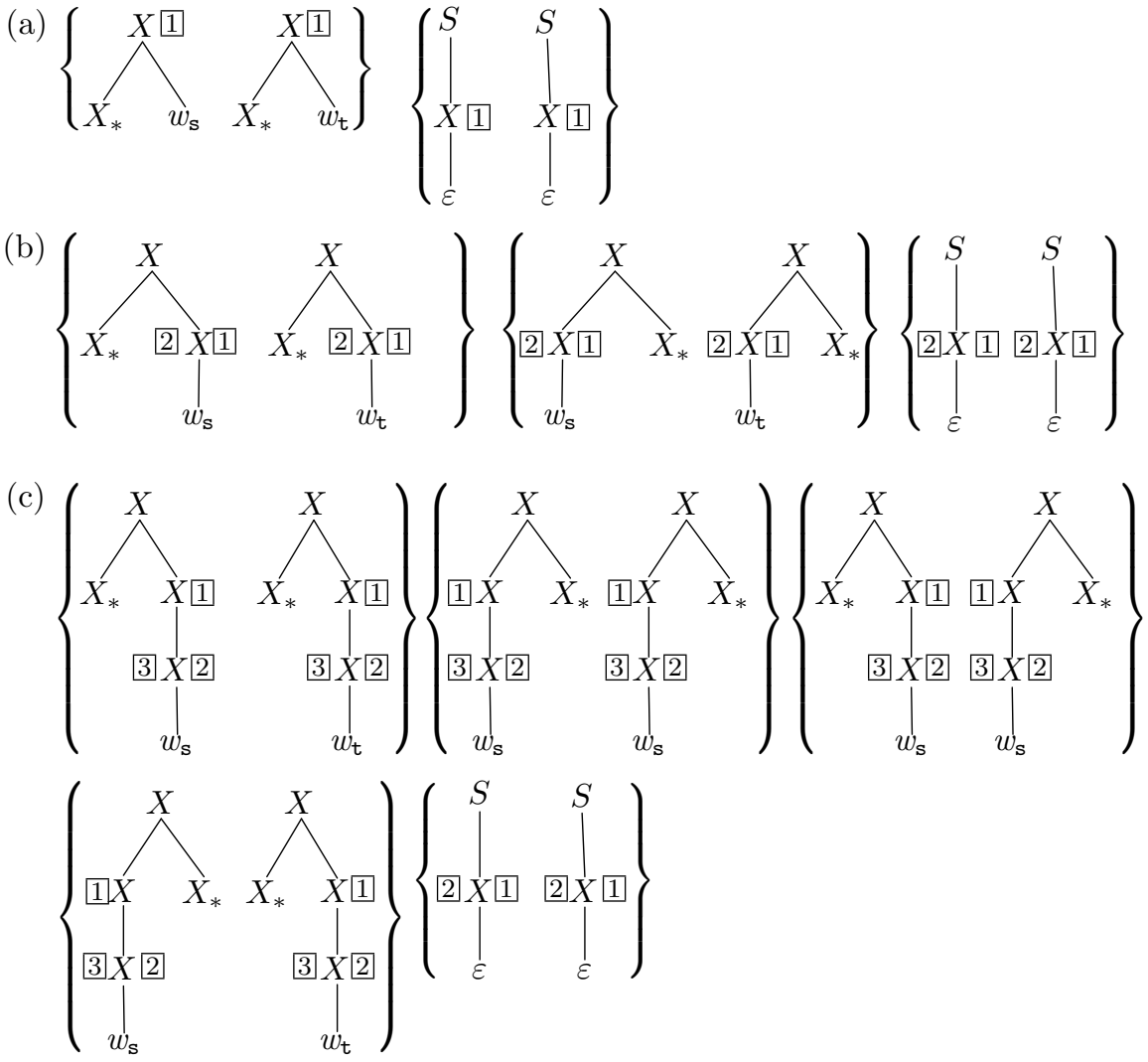


Figure 7.12: Possible canonical forms for synchronous TIG tree pairs: (a) the bigram model, (b) the extended bigram model, (c) our canonical tree pair form.

probability distributions will be specified. We do not have a bank of tree pairs to draw on for our elementary trees, so we must construct them. In addition, because we are going to iteratively refine the parameters, we need to make sure that the initial grammar is sufficiently general to capture any pair of input sentences. In order to do this, we define a canonical tree form.

Following Hwa [2001], we note that the very simple tree pairs shown in Figure 7.12(a) allow us to simulate a bigram model. Expanding to capture some of the benefit of tree structure, the trees can be modified to have their anchors on the left as well as the right and to have an additional adjunction node between the root and the anchor where both right and left trees can adjoin. This is depicted in Figure 7.12(b). Note that the simple modification of allowing both left and right trees results in four different types of tree pairs. Finally, we add an additional adjunction node, again following Hwa's empirical findings that two adjunction sites were required in the monolingual TIG case to handle the variety and number of modifiers in natural language data [Hwa, 2001]. This tree form is shown in Figure 7.12(c). Other canonical forms may be imagined as well. In order to ensure full coverage of the input strings we generate tree pairs for each pair of source and target language words that appear in the source and target dictionaries.

In addition to the shape of the trees, we also have to specify the links. Since these links determine both where and how many adjunctions may take place, their placement is quite important. Different types of links help to express different dependencies. In particular, links between two nodes on the same side express dependencies where the order of the dependent with respect to the anchors are the same in the two languages. For instance, if adjectives precede nouns in both languages we would want left-left links between noun nodes to express this dependency. Links between two nodes on different sides would correspondingly allow us to express dependencies where adjectives appear on different sides of the noun they modify in the two languages. Differences in the order of modifiers of the same lexical item can be expressed

by links that change level in the tree. Links to nodes closer to the anchor will result in the dependent words appearing closer to the anchor. This difference could be used to express differences in the order of modifiers, for instance different orderings for attachment of prepositional phrases. There are many possible combinations of links, even with our restrictions on link placement. In the experiments reported here, we use the links depicted in Figure 7.12(c).

In the E step we parse all of the sentences in the training corpus while maintaining expected counts. The inside probabilities are calculated simply by parameterizing the parser with the Inside semiring as described above. The outside probabilities are then calculated in a pass over the chart, this time working from the root of the start trees down to the leaves. The intuitive idea is that we work backwards, undoing each operation that was done to parse each sentence. That is, we work from a consequent item back to its antecedent. The outside probability of an antecedent item is the product of the outside probability of the consequent item, the inside probability of the other antecedent item, and the semiring value of the operation being undone.

In the M step we update the probabilities of all of the parameters by normalizing the counts produced in the E step received for each parameter.

7.2.5 Preliminary Empirical Results

As an initial empirical test of the formalism and algorithms described here, we chose a simple, artificial language problem: translation of arithmetic expressions from postfix to infix, from `ABA+*` to `A*(B+A)` for instance.

The expression language includes constants `A` and `B` and operators `+` and `*`. This

translation problem, though contrived, has several attractive properties as a simple test of syntax-aware MT techniques. First, it exhibits the type of hierarchical organization that we expect is difficult for non-syntax-aware translation systems. Second, the aligned corpora are easy to automatically generate, and translation correctness to automatically verify, even though, like natural language, there may be multiple correct target translations (differing in parenthesization) for a given source expression. Third, the need for extra parentheses in the target language forces differing lengths of source and target strings, so that this aspect of the translation formalism is exercised. Fourth, correctly aligned operators can be arbitrarily far apart in correct translations; in the terminology of IBM-style models, large distortions are manifest in translations, and these distortions are best characterized syntactically. Finally, the languages have very small vocabularies, which makes it possible to test the ability of our system in the absence of a fully optimized implementation.

We generated a corpus of aligned postfix and infix arithmetic expressions using a synchronous probabilistic expression-term-factor grammar that allows a single postfix expression to have several distinct infix translations that differ in the number and placement of parentheses, so long as sufficient parentheses are available to disambiguate properly according to the standard associativity and precedence of expressions. The inference rules used in the evaluation allow one tree in a tree pair to be anchored by ϵ, as in Section 7.2.2 and omit the use of unsynchronized tree pairs.[8] The full canonical form in Figure 7.12(c) was used, allowing (initially at least) any

[8]In practice this is done by using axioms that introduce paired nodes rather than unsynchronized trees but it is equivalent to requiring the Merge rule to apply immediately following the introduction of an item using the unsynchronized axioms.

Input expression	STIG Model Translation
B A B + B * +	B + (A + B) * B
A B A * + A *	(A + B * A) * A
A B B A + A * + +	A + B + ((B + A) * A)
B A A + A A + + + B *	(B + (A + A) + A + A) * B

Figure 7.13: Representative examples of correct translations of postfix arithmetic expressions to infix arithmetic expressions by our system.

symbol in the source language to translate as any symbol in the target language, and, through pairing with ϵ anchors, any symbol in either langauge to be inserted freely.

We trained the system on 411 aligned pairs of postfix and infix expressions with a maximum sentence length of 8 for the postfix expressions. We then tested the system on 90 test expressions without restriction on the length of the expressions, although most were not longer than 12 characters. The system produced correct infix expressions for 87 of the 90 test sentences, giving an error rate of 3.3%. Trained on 5000 pairs of aligned expressions, the system achieved 100% accuracy on the test set. Figure 7.13 shows representative examples of input expressions and their generated translations using the system trained on the small training set.

Of the three errors made by our system, two were relatively minor: in one of them a necessary set of parentheses around an addition is missing, the other is missing a right parenthesis. In the third erroneous translation it appears that the trained grammar was more seriously off track. Figure 7.14 shows the erroneous translations by our system.

As a baseline, we also trained GIZA++, a widely used implementation of an IBM-style system, on the same 411 training sentence pairs. The GIZA model trained on

Input expressions	Translation
A A A B + * + A A + *	(A * (A + B) + A) * A + A
A A B B * + +	A + A + (B * B
B A A * + B B + B * +	B + ((A + ((A)) * + B) * B

Figure 7.14: The three test expressions on which our MT system produced an incorrect result.

the small training set correctly translated 52 of 90 test sentences for an error rate of 42.2%. Unsurprisingly, GIZA performed quite poorly because of its lack of syntax awareness. To verify that the problem was not merely a lack of training data, we also trained GIZA on a training set consisting of 10,000 aligned sentence pairs to see if the additional training data would improve its performance. Again, the GIZA model trained on the large training set performed poorly, correctly translating only 53 of the 90 test sentences for an error rate of 41.1%. As expected, the GIZA models did well with the shorter, simpler sentences that did not require significant long-distance movement of any operators or parenthesization in the infix translation. Both models exhibited a tendency not to move constants or operators long distances and neither ever correctly parenthesized an infix expression. Figure 7.15 shows the output of GIZA on some sample expressions that our system translated correctly. Figure 7.16 shows the GIZA translations of the three sentences that our system translated incorrectly. In only one case did the GIZA system trained on 10,000 sentences produce a correct translation where our model erred.

Given the small size of the training set, the error rate for our system is quite low on the test data and demonstrates that our system appropriately learned to translate postfix arithmetic expressions to infix arithmetic expressions. Further, the

Input expression	GIZA translations	
	small training set	large training set
B A B + B * +	B A + B * B)	B * (B + A + B
A B A A A + + B * + +	A + B * A * A	A * (A + A *
A B B A + A * + +	A + B B A * A)	A + A * (A + B +
B A A + A A + + + B *	B A + A A + A) * B	B + A + A) * B

Figure 7.15: Results of GIZA on the sample sentences correctly translated by our system as shown in Figure 7.13.

Input expression	GIZA translations	
	small training set	large training set
A A A B + * + A A + *	((A + B * A * A)	A + A + A + A * B *
A A B B * + +	(A * B + B	A + A + B * B
B A A * + B B + B * +	B A * A + B + B * B)	B * (A + B + B * B) *

Figure 7.16: Translations produced by GIZA on the three sentences on which the STIG model produced an incorrect result.

large gap in quality and accuracy of translations between our system and the GIZA models shows that the capacity of our system to model hierarchical relationships with substitution does help our system to correctly translate sentences that are difficult for an IBM-style model.

The system used in these experiments used relatively aggressive thresholding. Even so, training took an hour four 411 sentence pairs. By contrast, GIZA took only seconds to train on the same data. Our system run without the thresholding and optimizations achieved a similar error rate to the optimized system but took approximately 30 hours to train. Since implementing the thresholding and other optimizations we have been able to train our system on 5000 pairs of arithmetic expressions. Training took approximately 12 hours. With this greater volume of

training data our system achieved 100% accuracy on the test set.

7.3 Syntactically-Motivated Trees

The statistically induced substrate is intended to be hybridized by the inclusion of syntactically-motivated trees. In this section we explore possible methods of generating syntactically-motivated trees for inclusion in the grammar.

7.3.1 Bilingual Dictionaries

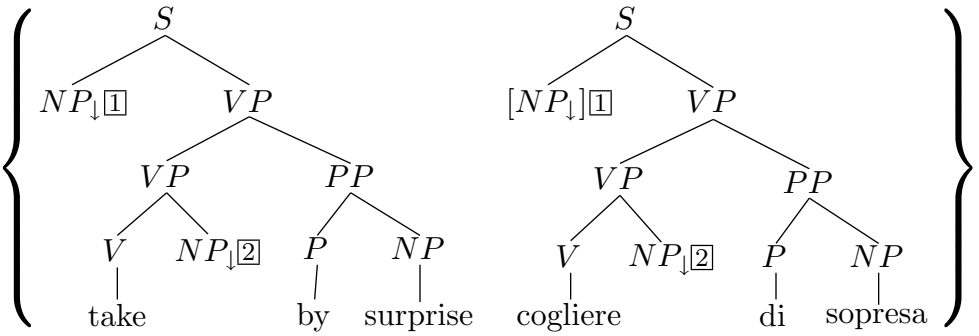


Figure 7.17: A tree pair that might be extracted from the bilingual dictionary entry ⟨*take* SB *by surprise / cogliere* QN *di sorpresa*⟩. Note that the link between the two object noun phrases marked for substitution comes directly from the variables in the dictionary entry. The internal syntactic structure as well as the linked positions for the subjects of the verbs would be generated by parsing and a basic verb template. The subject position in the Italian is bracketed to indicate that it is optional.

As argued in Shieber [2007], bilingual dictionaries offer a substantial repository of construction-based aligned trees that might be extracted. Additionally, for construction-based MT, reconstruction of tree alignments from data is much more difficult than for phrase-based MT, and hence extracting them from a dictionary becomes quite

appealing. We propose to repurpose bilingual dictionaries as banks of aligned trees for construction-based MT. Using simple parsing techniques and the already present variable notation, STIG tree pairs can be constructed directly and automatically from dictionary entries. Consider the example ⟨*take* SB *by surprise / cogliere* QN *di sorpresa*⟩ taken from the HarperCollins Italian College Dictionary (HCICD) [Clari and Love, 1995]. Figure 7.17 shows two tree pairs that could be extracted from this dictionary entry. Obtaining a processable copy of a bilingual dictionary presents a challenge. Corpus development efforts, such as EuroWordNet[9], provide one easily accessible place to begin. We also intend to pursue direct agreements with dictionary publishers to obtain data from sources such as HCICD.

7.3.2 Harvesting from a Treebank

Although generation of tree alignment is a challenging task to automate, several lines of research have made inroads that we believe will make it possible to extract aligned trees from an aligned treebank of parsed sentence pairs. Intuition as to how this might be done on a particular example helps to illustrate the idea. Consider the sentence pair *The dog took Nina by surprise/Il cane ha colto Nina di sopresa*. Using word and small phrase alignments (for instance, *the dog/il cane*, *Nina/Nina*, and *by surprise/di sopresa*) we can identify places where the parse tree might be factorized. Simple syntactic information, such as that noun phrases are likely to serve as arguments and combine by substitution helps to decide how to slice the tree. Syntactic patterns such as the repeating VP nodes provide a suggestion of

[9] <http://www.illc.uva.nl/EuroWordNet/>

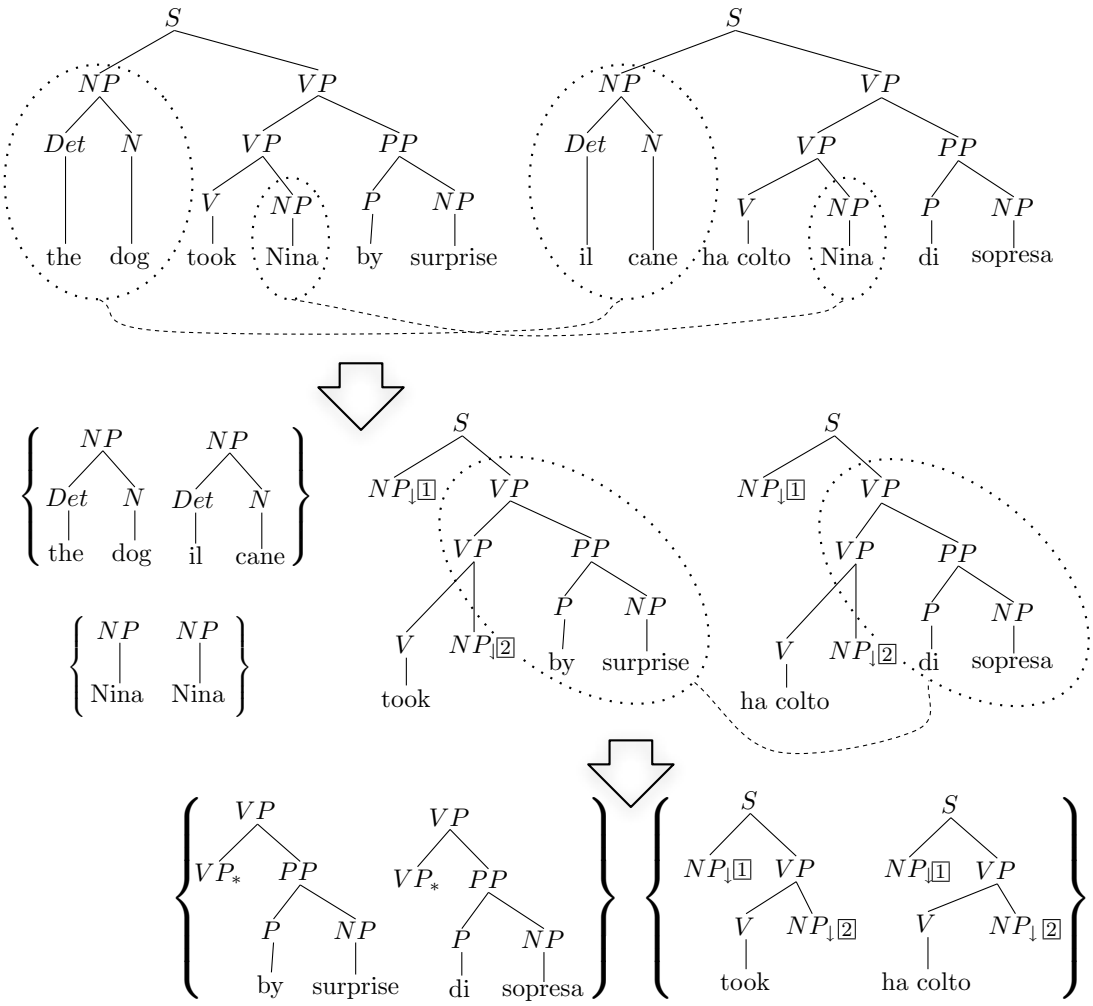


Figure 7.18: Possible tree extraction from the parse trees for the sentence pair *The dog took Nina by surprise./Il cane ha colto di sopresa Nina.*

where an auxiliary tree might be extracted. This could result in a set of tree pairs being extracted as shown in Figure 7.18. Note that while the extraction can break the tree down into these small pieces, it can also retain tree pairs that are larger fragments of the sentence. This would result in retention of the desired tree in which the prepositional phrase remains combined with the verb as well as the two smaller tree pairs in which the verb and prepositional phrase have been separated.

Groves et al. [2004] present an automated algorithm for aligning subtrees of pairs of parse trees of sentences that are translations of each other. The algorithm relies on a small amount of syntactic information and a word-alignment that is internal to the trees. It produces the equivalent of STAG elementary tree pairs that could be added to our substrate grammar to hybridize it. Although the alignments produced by this method are not as good as manual alignments, the results presented in their work demonstrate that they are usable. In our system in which we propose an additional round of statistical training using the syntactically motivated trees obtained in this step, tree pairs that are misaligned are likely to be infrequently useful and therefore to be assigned low probability of adjoining into other tree pairs.

Another related line of work on which we expect to draw is the body of work on induction of TAG grammars from a treebank exemplied by Chen and Shanker [2004a]. Working from a treebank of natural language sentences, this work extracts a TAG grammar using syntactic constraints. The work relies heavily on syntactically motivated rules for identifying elementary structures within the derived trees. Although it is focused solely on monolingual sentences, we would seek to apply its insights to extracting aligned trees.

Finally, our own work on the factorization of synchronous TAGs (Chapter 4) explores breaking down a pair of trees into smaller fragments that can be used to reconstruct the original tree. This work focuses on reducing the parsing complexity of an input synchronous TAG and contemplates working to reduce the size of the elementary trees of a grammar rather than working with complete parse trees. However, there is nothing to prevent the application of the methodology to a complete parse

tree. Additionally, the insight of this work into the effect of the location of alignments within elementary trees on parsing complexity will be extremely valuable in ensuring that the trees used to hybridize the system do not increase the parsing complexity beyond acceptable bounds.

7.4 Combining the Two Grammars

Two methods seem promising for initial exploration of combining the statistically induced substrate PSTIG grammar and the STIG elementary trees harvested from bilingual dictionaries and treebanks. A first option is to add the additional trees produced in the harvesting stage directly into the substrate grammar produced in the statistical induction stage. Since the harvested trees have no accompanying adjunction probabilities, some form of retraining of the grammar would have to be done after they are combined. One possibility is to randomly or equally assign adjunction counts to the harvested trees (and also randomly or equally generate counts for their adjunction parameters). Once their parameters and the probability of their adjunction into the substrate trees are set, the expectation maximization algorithm can be run again to retrain the parameters. A second possibility is to train both the normal form and the harvested trees together as a single grammar in a single step.

A second, simpler option is to use the statistical substrate grammar as a backoff for the grammar generated by harvesting. That is, knowing that the harvested grammar will necessarily be a low coverage grammar but is likely to capture some important construction-based translations, the system might seek to produce a translation using only trees from the harvested grammar and only resort to the substrate grammar when

all or a part of a parse cannot be obtained using the harvested grammar.

7.5 Conclusion

This chapter presents a research program for building a synchronous-grammar-based statistical machine translation system that, through hybridization, captures the benefits both of traditional SMT systems and hand-coded translation systems. The initial work to create a statistically-induced substrate based in synchronous TIG is presented here with results and a roadmap for the incorporation of syntactically-motivated trees to improve performance.

Chapter 8

Conclusion

In this dissertation I have argued for the sufficiency of synchronous grammar formalisms to express complex relationships between related natural language inputs such as the syntax and semantics of a language or the syntax of two different languages. The crux of this issue is whether it is possible to find a base formalism to synchronize that is at once sufficiently flexible and expressive to capture the complex structure of each of the linguistic inputs and their relationship and sufficiently simple that the synchronization provides meaningful constraint and processing can be accomplished reasonably efficiently. I chose to focus on formalisms in the Tree-Adjoining Grammar family when seeking a base formalism to synchronize both for historical reasons and because it is particularly well suited to modeling syntactic structure.

The first part of the thesis explored the question of processing efficiency for grammar formalisms in the multicomponent and synchronous TAG family. It proved novel results showing that Tree-Local MCTAG (and everything above it in the multicomponent TAG hierarchy) is hard to process under most conditions. It also provided

various algorithms to help mitigate the potential inefficiency of working with a TL-MCTAG or closely related formalism with preprocessing algorithms that optimize the grammars and efficient parsing algorithms. This part of the thesis narrowed the focus of the search for a good base formalism to those that are at least no less efficient to process than TL-MCTAG and raised the question of whether even TL-MCTAG will be too inefficient to process in practice. Empirical questions about how much actual input grammars can be optimized and how inefficient recognition will be in practice remain.

In addition, in the first part of the thesis I presented an algorithm for optimizing a synchronous TAG because, as I discussed in Chapter 4, all synchronous grammars where the base formalism is at least as expressive as CFG are NP-hard to recognize. The hardness of processing synchronous grammars still stands as a challenge to implementations of systems based on a synchronous grammar framework. The work in this thesis treated all linguistic inputs equally rather than treating one input as a source and the other as a target. However, in practice most applications will be directional. In translation we are generally given a sentence of one language and asked to produce a sentence of another. In the domain of semantics it is easy to imagine a system in which we receive a surface sentence or even a syntactic parse tree as input and are asked to produce the semantics. A generation application might begin with the semantics and produce the syntax. Given this directionality, research into algorithms for using the information that is given on one side of a synchronous grammar to constrain the search space for a synchronous parse should be fruitful.

Chapters 5 and 6 in the second part of the thesis applied synchronous TL-MCTAG

and synchronous LDV-TAG to the analysis of the interface between syntax and semantics. By covering a wide range of phenomena these chapters showed that a synchronous grammar can capture the highly divergent structures of syntax and semantics and their interface. Although the range of phenomena covered is broad, it only scratches the surface of the English language both in breadth and depth. This work, therefore, might serve as the kernel to an implementation of a synchronous LDV-TAG for English syntax and semantics that could over time be built into a broad coverage system. Rather than presenting finished research, Chapter 7 laid out a research program for using synchronous grammars in machine translation. Following this program through to an implementation would further support the thesis of this dissertation.

Taken individually, the results in this dissertation are both useful and suggestive of avenues for future research. Taken collectively, they demonstrate the clear but not obvious potential of synchronous grammars to efficiently model complex linguistic relationships.

Bibliography

Anne Abeillé and Yves Schabes. Parsing idioms in tree adjoining grammars. In *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, 1989.

Anne Abeillé, Yves Schabes, and Aravind K. Joshi. Using lexicalized tree adjoining grammars for machine translation. In *Proceedings of the Thirteenth International Conference on Computational Linguistics (COLING '90)*, Helsinki, August 1990.

Alfred V. Aho and Jeffrey D. Ullman. Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*, 3(1):37–56, 1969.

J.K. Baker. Trainable grammars for speech recognition. In D.H. Klatt and J.J. Wolf, editors, *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550, 1979.

P.F. Brown, S.A. Della Pietra, V.J. Della Pietra, and R.L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.

John Chen and Vijay K. Shanker. Automated extraction of tags from the penn treebank. *New Developments in Parsing Technology*, pages 73–89, 2004a.

John Chen and Vijay K. Shanker. Automated extraction of TAGs from the penn treebank. *New developments in parsing technology*, pages 73–89, 2004b.

David Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 33 (2):201–228, 2007.

David Chiang and Owen Rambow. The hidden TAG model: synchronous grammars for parsing resource-poor languages. In *Proceedings of the 8th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 8)*, pages 1–8, 2006.

David Chiang and Tatjana Scheffler. Flexible composition and delayed tree-locality. In *The Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+9)*, 2008.

Michela Clari and Catherine E. Love, editors. *HarperCollins Italian College Dictionary*. Harper-Collins Publishers, Inc., New York, NY, 1995.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

J. Earley. *An efficient context-free parsing algorithm*. PhD thesis, University of California, Berkeley, California, 1970.

Robert Frank. Reflexives and tag semantics. In *The Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+9)*, T ubingen, Germany, June 2008.

Robert Frank. *Syntactic Locality and Tree Adjoining Grammar: Grammatical Aquisition and Processing Perspectives*. PhD thesis, University of Pennsylvania, 1992.

M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman and Co., New York, NY, 1979.

Daniel Gildea, Giorgio Satta, and Hao Zhang. Factoring synchronous grammars by sorting. In *the International Conference on Computational Linguistics/Association for Computational Linguistics (COLING/ACL-06) Poster Session*, 2006.

Joshua Goodman. Semiring parsing. *Computational Linguistics*, 25(4):573–605, 1999.

S.L. Graham, M.A. Harrison, and W.L. Ruzzo. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2:415–462, 1980.

Declan Groves, Mary Hearne, and Andy Way. Robust sub-sentential alignment of phrase-structure trees. In *In Proceedings of the 20th International Conference on Computational Linguistics (COLING-04)*, pages 1072–1078, 2004.

Chung-Hye Han. A tree adjoining grammar analysis of the syntax and semantics of it-clefts. In *Proceedings of the 8th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 8)*, pages 33–40, Sydney, Australia, 2006a.

Chung-Hye Han. Pied-piping in relative clauses: Syntax and compositional semantics based on synchronous tree adjoining grammar. In *Proceedings of the 8th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 8)*, pages 41–48, Sydney, Australia, 2006b.

Chris Hankin. *An Introduction to Lambda Calculi for Computer Scientists*. King's College Publications, 2004.

Jerry Hobbs and Stuart M. Shieber. An algorithm for generating quantifier scopings. *Computational Linguistics*, 13(1-2):47–63, 1987.

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

A. K. Joshi. An introduction to tree adjoining grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins, 1987.

A. K. Joshi, L. S. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1), 1975.

Aravind K. Joshi. *How Much Context-Sensitivity Is Necessary for Characterizing Structural Descriptions—Tree-Adjoining Grammar*. Cambridge University Press, 1985.

Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, pages 69–124. Springer, 1997.

Aravind K. Joshi, Laura Kallmeyer, and Maribel Romero. Flexible composition in LTAG: Quantifier scope and inverse linking. In Ielka van der Sluis Harry Bunt and Roser Morante, editors, *Proceedings of the Fifth International Workshop on Computational Semantics IWCS-5*, pages 179–194, Tilburg, January 2003.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice-Hall, 2000.

Yuichi Kaji, Ryuchi Nakanishi, Hiroyuki Seki, and Tadao Kasami. The universal recognition problems for multiple context-free grammars and for linear context-free rewriting systems. *IEICE Transactions on Information and Systems*, E75-D (1):78–88, 1992.

Yuichi Kaji, Ryuchi Nakanishi, Hiroyuki Seki, and Tadao Kasami. The computational complexity of the universal recognition problem for parallel multiple context-free grammars. *Computational Intelligence*, 10(4):440–452, 1994.

Laura Kallmeyer. A declarative characterization of different types of multicomponent tree adjoining grammars. In Andreas Witt Georg Rehm and Lothar Lemnitzer, editors, *Datenstrukturen für linguistische Ressourcen und ihre Anwendungen*, pages 111–120, 2007.

Laura Kallmeyer and Aravind K. Joshi. Factoring predicate argument and scope semantics: Underspecified semantics with LTAG. *Research on Language and Computation*, 1:3–58, 2003.

Laura Kallmeyer and Maribel Romero. LTAG semantics with semantic unification. In *Proceedings of the 7th International Workshop on Tree-Adjoining Grammars and Related Formalisms (TAG+7)*, pages 155–162, Vancouver, May 2004.

Laura Kallmeyer and Maribel Romero. Reflexives and reciprocals in ltag. In Harry Bunt Jeroen Geertzen, Elias Thijsse and Amanda Schiffrin, editors, *Proceedings of the Seventh International Workshop on Computational Semantics ICWS-7*, pages 271–282, Tilburg, January 2007.

Laura Kallmeyer and Tatjana Scheffler. LTAG analysis for pied-piping and stranding of WH-phrases. In *Proceedings of TAG+7*, pages 32–39, Vancouver, British Columbia, May 2004.

T. Kasami. An efficient recognition and syntax algorithm for context-free languages. Technical Report AF-CRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA, 1965.

K. Lari and S.J. Young. Applications of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 5:237–257, 1991.

I. Dan Melamed. Multitext grammars and synchronous parsers. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 79–86, 2003.

I. Dan Melamed. Statistical machine translation by parsing. Technical Report 04–024, Proteus Project, 2004.

I. Dan Melamed, Giorgio Satta, and Ben Wellington. Generalized multitext grammars. In *Proceedings of the 42nd Annual Conference of the Associate for Computational Linguistics (ACL-04)*, 2004.

Rebecca Nesson and Stuart Shieber. Efficiently parsable extensions to tree-local multicomponent tag. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, Boulder, CO, 2009.

Rebecca Nesson and Stuart M. Shieber. Extraction phenomena in synchronous TAG syntax and semantics. In *Proceedings of Syntax and Structure in Statistical Translation (SSST)*, Rochester, NY, April 2007.

Rebecca Nesson and Stuart M. Shieber. Synchronous vector tag for syntax and semantics: Control verbs, relative clauses, and inverse linking. In *The Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+9)*, T ubingen, Germany, June 2008.

Rebecca Nesson and Stuart M. Shieber. Simpler TAG semantics through synchronization. In *Proceedings of the 11th Conference on Formal Grammar*, Malaga, Spain, 29–30 July 2006. URL `http://www.eecs.harvard.edu/ shieber/Biblio/Papers/Nesson-2006-SSS.pdf`.

Rebecca Nesson, Alexander Rush, and Stuart M. Shieber. Induction of probabilistic synchronous tree-insertion grammars. Technical Report TR-20-05, Harvard University, 2005.

Rebecca Nesson, Stuart M. Shieber, and Alexander Rush. Induction of probabilistic synchronous tree-insertion grammars for machine translation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA 2006)*, Boston, Massachusetts, 8-12 August 2006. URL `http://www.eecs.harvard.edu/ shieber/Biblio/Papers/Nesson-2006-IPS.pdf`.

Rebecca Nesson, Giorgio Satta, and Stuart Shieber. Optimal k-arization of synchronous tree-adjoining grammar. In *the Association for Computational Linguistics (ACL-2008)*, Columbus, OH, June 2008a.

Rebecca Nesson, Giorgio Satta, and Stuart M. Shieber. Complexity, parsing, and factorization of tree-local multi-component tree-adjoining grammar. Technical report, Harvard University, 2008b.

D. Prescher. Inside-outside estimation meets dynamic EM. In *In Proceedings of IWPT-2001*, 2001.

Owen Rambow. *Formal and computational aspects of natural language syntax*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1994.

Maribel Romero, Laura Kallmeyer, and Olga Babko-Malaya. Ltag semantics for questions. In *Proceedings of the 7th International Workshop on Tree-Adjoining Grammars and Related Formalisms (TAG+7)*, pages 186–193, Vancouver, May 2004.

Giorgio Satta. Recognition of linear context-free rewriting systems. In *Proceedings of the 10th Annual Meeting of the Association for Computational Linguistics (ACL92)*, pages 89–95, Newark, Delaware, 1992.

Giorgio Satta and Enoch Peserico. Some computational complexity results for synchronous context-free grammars. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT05/EMNLP05)*, pages 803–810, Vancouver, British Columbia, 2005.

Yves Schabes and Stuart M. Shieber. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124, 1993.

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

Yves Schabes and Richard C. Waters. Tree insertion grammar: A cubic time, parsable formalism that lexicalizes context-free grammars without changing the trees produced. *Computational Linguistics*, 21(4):479–512, 1995.

William Schuler, David Chiang, and Mark Dras. Multi-component tag and notions of formal power. In *Proceedings of the Thirty-Eighth Annual Meeting of the Association for Computational Linguistics (ACL'00)*, pages 448–455, Hong Kong, China, 2000.

H. Seki, T. Matsumura, M. Fujii, and T. Kasami. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229, 1991.

Stuart M. Shieber. Probabilistic synchronous tree-adjoining grammars for machine translation: the argument from bilingual dictionaries. In Dekai Wu and David Chiang, editors, *Proceedings of the Workshop on Syntax and Structure in Statistical Translation*, Rochester, NY, April 2007.

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

Stuart M. Shieber and Yves Schabes. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124, 1994.

Stuart M. Shieber and Yves Schabes. Synchronous tree adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING '90)*, Helsinki, August 1990.

Stuart M. Shieber, Yves Schabes, and Fernando Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:503–512, 1994.

Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36, July–August 1995. Also available as cmp-lg/9404008.

S. Sippu and E. Soisalon-Soininen. *Parsing Theory: Languages and Parsing*. Springer-Verlag, Berlin, Germany, 1988.

Anders Søgaard, Timm Lichte, and Wolfgang Maier. On the complexity of linguistically motivated extensions of tree-adjoining grammar. In *Recent Advances in Natural Language Processing 2007*, 2007.

C. A. Tovey. A simplified NP-complete satisfiablity problem. *Discrete Applied Mathematics*, 8(1):85–90, 1984.

Kurt VanLehn. Determining the scope of English quantifiers. Technical Report 483, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1978.

K. Vijay-Shanker. A study of tree-adjoining grammars. PhD Thesis, Department of Computer and Information Science, University of Pennsylvania, 1987.

K. Vijay-Shanker and Aravind K. Joshi. Some computational properties of tree-adjoining grammars. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 82–93, 1985.

David Weir. Characterizing mildly context-sensitive grammar formalisms. PhD Thesis, Department of Computer and Information Science, University of Pennsylvania, 1988.

Sean Williford. Application of synchronous tree-adjoining grammar to quantifier scoping phenomena in English. Undergraduate Thesis, Harvard College, 1993.

D.H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, 1967.

Hao Zhang and Daniel Gildea. Factorization of synchronous context-free grammars in linear time. In *NAACL Workshop on Syntax and Structure in Statistical Translation (SSST)*, 2007.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. Synchronous binarization for machine translation. In *Proceedings of the Human Language Technology Conference/North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*, 2006.