A.K. Joshi

CHARACTERIZING MILDLY CONTEXT-SENSITIVE GRAMMAR FORMALISMS

David J. Weir

MS-CIS-88-74
LINC LAB 132

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

September 1988

Acknowledgements: This research was supported in part by DARPA grant NOOO14-85-K-0018, NSF grants MCS-8219196-CER, IRI84-10413-AO2, MCS 82-07294, DCR 84-10413 and U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027.

Characterizing Mildly Context-Sensitive Grammar Formalisms

David J. Weir

A DISSERTATION

IN

COMPUTER AND INFORMATION SCIENCE

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy.

1988

[illegible]

Supervisor of Dissertation

[illegible]

Graduate Group Chairperson

Acknowledgments

I am very grateful to my thesis advisor Aravind Joshi. I am extremely lucky to have been able to benefit from his deep insight into this field. His advice has been crucial throughout my dissertation research. I am indebted to Jean Gallier, Tony Kroch, Mike Palis, Mark Steedman, Bonnie Webber, and Dawn Griesbach for their help, especially Mark for his patience in explaining Categorial Grammars to me, and Tony for giving me some understanding of what it really means to use a grammar formalism to express a linguistic theory. My friend and collaborator Vijayashanker has had a considerable influence on this research, and has been a continuous source of valuable ideas, many of which appear in this thesis. I would also like to thank Tomas Isakowitz, Chick Kozloff, Wayne Snyder, and my other friends in the CIS department for all of their kindness. I thank my mother for her support, and my wonderful wife Jane for the encouragement she has given me, and for making the duration of this dissertation so enjoyable.

This work was partially supported by NSF grants MCS-82-19116-CER, MCS-82-07294, and DCR-84-10413.

Abstract

Characterizing Mildly Context-Sensitive Grammar Formalisms

David J. Weir

Supervisor: Aravind K. Joshi

This thesis involves the study of formal properties of grammatical formalisms that are relevant to computational linguists. The formalisms which will receive the most attention share the property that they are highly restricted in their generative power. Recent research suggests that Context-Free Grammars (CFG's) lack the necessary expressive power on which to base a linguistic theory. This has led computational linguists to consider grammatical formalisms whose generative power exceeds CFG's, but to only a limited extent. We compare a number of formalisms on the basis of their weak generative capacity, as well as suggesting ways in which they can be compared on the basis of their strong generative capacity. In particular, we consider properties of their structural descriptions (or tree sets); and the types of dependencies (nested, crossed, etc.) that can be exhibited by each formalism.

Several formalisms that are notationally quite different (Tree Adjoining Grammars, Head Grammars, and Linear Indexed Grammars) have been shown to be weakly equivalent. We show that Combinatory Categorial Grammars are weakly equivalent to these formalisms. The class of languages generated by these formalisms can be thought of one step up from CFG's, and we describe a number of progressions that illustrate this.

The string languages generated by TAL's, HL's, CCL's and LIL's exhibit limited crossed-serial dependencies in addition to those produced by Context-Free Grammars (nested and serial dependencies). By formalizing these crossed-serial depen-

dencies and their relationship with the nested dependencies produced by CFG's we define an infinite progression of formalisms.

Our work on structural descriptions leads us to characterize a class of formalisms called Linear Context-Free Rewriting Systems (LCFRS's), which includes a wide range of grammatical formalisms with restricted power. The systems in this class have context-free derivations, and simple composition operations that are linear and nonerasing. We prove that all members of this family generate only semilinear languages that can be recognized in polynomial time.

Contents

List of Figures

Chapter 1

Introduction

This dissertation involves the study of formal properties of grammatical formalisms, with particular emphasis on those properties that are relevant to computational linguists. The formalisms which will receive the most attention share the property that they are highly restricted in their generative power. We begin by very briefly explaining the rationale behind the use of constrained grammar formalisms.

Chomsky [18, 16, 15] introduced the notion of Context-Free Grammars (CFG's), yet he argued [18, 17] that natural languages fell well outside the class of Context-Free Languages (CFL's). For some time after this, relatively little attempt was made to develop fully explicit grammatical theories that could be presented in a mathematically precise grammar formalism. However, recently a number of theories have been developed using grammar formalisms that have reasonably clear definitions, and this has led to study of their mathematical and computational properties (for example, [8]). Several researchers have made use of grammar formalisms that have close to context-free power, questioning the validity of arguments that considerably more power than CFG's is needed to describe natural languages (see, for example, [25, 64, 62]).

Gazdar et al [26] developed a detailed linguistic theory called Generalized Phrase-Structure Grammars (GPSG), using a formalism whose weak generative power was equivalent to CFG's. There is linguistic evidence that certain natural languages are not context-free [12, 74, 20]. Several other formalisms with slightly more power than CFG's that include these cases have also been introduced, and used to implement various linguistic theories: Head Grammars [62], an extension of CFG's; Combinatory Categorial Grammars [2, 78, 75, 76], an extension of Classical Categorial Grammars [5] (a formalism known to be weakly equivalent to CFG's [7]); and Tree Adjoining Grammars [42, 36], a tree manipulating system. This investigation of the extent to which constrained grammar formalisms can describe aspects of natural languages leads to a greater understanding of where the class of natural languages falls.

There is a second benefit to be gained from this enterprise; that of allowing the implementation of more concise linguistic theories. Although these formalisms have constrained generative power, not all of the grammars expressible in these formalisms generate plausible natural languages. Thus, an adequate linguistic theory that used one of these formalisms must also include a number of rules, or linguistic stipulations, concerning which grammars are well-formed. Use of a completely unconstrained grammar formalism leaves the entire linguistic theory in the form of such stipulations. One of the advantages that may arise from use of a more highly constrained grammar formalism (when the constraints take the appropriate form) is that fewer stipulations are needed, since some of them will fall out of the constraints in the grammar and other stipulations. This has been demonstrated in the case of the TAG formalisms in several ways, as discussed in [49, 48]. In [49] it is shown that if certain well-formedness constraints on simple sentences are stipulated, in particular that they do not begin with multiple *wh's*, then extraction from *wh*-islands is predicted to be ungrammatical. [48] extends this result by giving an account of several violations of subjacency using a slight variant of the original TAG formalism. These finding suggest that the effect of subjacency falls out of the well-formedness of the basic structures of the grammar and the adjunction operation. In [49] it is also shown that by giving a simple statement of the Empty Category Principle as a well-formedness constraint on simple sentences, it is not necessary to independently stipulate the so-called Condition on Extraction Domains.

Having given some justification for the use of a grammar formalism with restricted expressive power, we turn to the issue of identifying what kinds of constraints are appropriate. In using a grammatical formalism, linguists wish to capture some of their intuitions about linguistic constraints through limitations in the descriptive power of the formalism. To make informed choices about suitable formalisms, they require that formalisms be characterized in a way that relates formal properties to linguistic intuitions. There is a need to bridge the gap between the kinds of properties that are typically used to characterize a formalism mathematically, and those requirements of a formalism that are meaningful and relevant to a linguist.

An example of such an attempt involves the so-called "constant growth property" [36, 11].

> L is **constant growth** if there is a constant c_0 and a finite set of constants C such that for all $w \in L$ where $|w| > c_0$ there is a $w' \in L$ such that $|w| = |w'| + c$ for some $c \in C$.

This mathematical property is intended to be an approximate characterization of the linguistic intuition that sentences of any natural language are built from a finite set of clauses of bounded structure using certain simple linear operations. While the constant growth property is too weak to capture this intuition fully, since it refers only to strings lengths, it represents an interesting attempt to formalize a linguistic intuition. The slightly stronger property of semilinearity may come closer, but is still only an approximation of what is really intended.

The notion of constant growth, together with polynomial parsability and the ability to produce "limited" cross-serial dependencies, are three conditions that Joshi [36] used to give a approximate characterization of a class of **mildly context-sensitive grammar formalisms.** This characterization is a useful first step in evaluating a formalism's suitability. However, with the exception of the condition on dependencies (which is not defined precisely), this characterization is solely in terms of string languages, or *weak* generative capacity. This is also true of almost all formal characterizations of grammar formalisms. While weak generative capacity is important, it would be useful to be able to formalize other aspects of a formalism's descriptive power that may not necessarily be expressible merely as properties of the string languages. Such aspects of a formalism are often referred to as its *strong* generative capacity. The problem with comparing the strong generative capacity of different formalisms and making general statements about how the strong generative capacity should be limited is that such criteria should be applicable to a range of radically different systems. Such criteria have been difficult to develop as, for example, the objects comprising the grammar (e.g., productions or trees), and the structural descriptions (e.g., trees, graphs) could be very different notationally. Part of the work described in this thesis is an attempt to develop criteria for evaluating the strong generative capacity of grammar formalisms. This involves elaborating

Joshi's criteria, and applying them to a range of grammar formalisms.

We now give an outline of this document, and show how each of the chapters are related.

1.1 Outline

Chapter 2 contains the definitions of several grammar formalisms: Context-Free Grammars, Head Grammars, Tree Adjoining Grammars, Indexed Grammars, Linear Indexed Grammars, and Multicomponent Tree Adjoining Grammars. Particular attention is paid to a comparison of the class of structural descriptions or tree sets that they can produce. We focus on their derivational process as reflected by their derivation trees and find that a number of the formalisms have similar properties. In considering the derivational process of a variety of formalisms that are mildly context-sensitive, and comparing them with others that are not, we may have come closer to formalization of the linguistic intuition behind the constant growth property.

The discussion of tree sets in Chapter 2 points to the existence of a range of new mildly context-sensitive formalisms. In Chapter 3 we define various progressions of grammar formalisms, and associated progression of string automata that appears to form natural progressions from Regular Languages to CFL's to TAL's, and beyond. In an attempt to understand better the notion of "limited" crossed dependencies, we investigate how to formalize string dependencies. We do this by describing single languages exhibiting all of the dependencies produced by the formalism, and again defining a progression of language classes. Each of the different progressions that are described in this chapter arise from the ability to formalize certain aspects of the strong generative capacity of previously defined formalisms. We can then maintain what appear to be the important similarities while generalizing certain differences.

In Chapter 4 we define a class of grammar formalisms that appears to resemble the class of mildly context-sensitive Grammar Formalisms. This class contains several of the formalisms considered in Chapter 2, and Chapter 3. We give details of how grammars of various formalisms can be expressed within this framework. We then consider properties of the string languages of members of this class, showing that they are semilinear languages, and that they are a proper subclass of the languages recognizable in polynomial time. Much of the contents of this chapter were first presented in [86].

Chapter 5 focuses on Combinatory Categorial Grammars. A version of Categorial Grammars is described that is shown to be weakly equivalent to Tree Adjoining Grammars, Head Grammars, and Linear Indexed Grammars. We discuss two extensions to this system and show that they increase its generative power. At the end of this chapter we briefly examine the tree sets produced by CCG's, and consider some issues associated with the representation of their derivations.

Chapter 6 summarizes the main contributions of this thesis and suggests topics of further research that arise from this work.

Chapter 2

Grammar Formalisms and Tree Sets

The study of the relationship between different grammar formalisms has almost always been in terms of weak generative capacity. While this has been valuable, our goal in this chapter is compare aspects of formalisms that have to do with the more important property of "strong" generative capacity. We consider a number of formalisms and for each we examine the structural descriptions that are produced, and focus particularly on the derivation process. The structural descriptions produced by CFG's take the form of phrase-structure trees. These trees provide an analysis of the sentence on the frontier, showing how it is decomposed into its constituents. Not all of the formalism that we will be discussing generate such structural descriptions. We consider tree sets that have annotated nodes, in which the yield of the tree can not simply be read off the frontier in the usual way. Some of the systems that we discuss in this chapter are not string manipulating formalisms. In some cases the structures being manipulated at the object level are themselves trees. In such cases, two tree sets can be associated with a grammar: the object level tree set that the grammar generates, and set of derivation trees that encode the derivation process.

We consider properties of the tree sets associated with CFG's, Head Grammars (HG's), Tree Adjoining Grammars (TAG's), Indexed Grammars (IG's), Linear Indexed Grammars (LIG's), and Multicomponent TAG (MCTAG's). We examine both the complexity of the paths of trees in the tree sets, and the kinds of dependencies that the formalisms can impose between paths: two properties that are not only linguistically relevant, but also appear to have computational importance. We find that the different formalisms produce a variety of families of tree sets. However, it is striking that several of the formalisms appear to share certain common properties. These similarities become clear when examining the derivation trees produced by each formalism. Derivation trees are very useful since they help us overcome the fact that we are comparing formalisms that are notationally very different. Derivation trees are very abstract structures that depend only indirectly on the details of the formalism's composition operation, and the structures being manipulated.

As we show, a number of the formalisms, with a fairly wide range of weak generative power, can be grouped together as having identically structured derivation tree sets, derivation tree sets that are similar to those of CFG's. All of the differences between the formalisms arise from the way in which their derivation trees must be interpreted in order to determine the structures that were derived in the derivation. This close relationship between some of the formalisms suggests that by generalizing the notion of context-freeness in CFG's, we can define a class of grammatical formalisms that manipulate more complex structures. In Chapter 4, we outline how such a family of formalisms can be defined, and show that like CFG's, each member possesses a number of desirable properties: in particular, the constant growth property and polynomial recognizability. [86] includes some of the material in this chapter.

The first characteristic of tree sets that we will be examining is the path set of a tree set.

Definition 2.0.1 The **path set** $\mathcal{P}(\gamma)$ of a labeled tree γ is the set of strings that label some path from the root to a leaf node of γ. It can be defined by induction as follows.

- If γ consists of a single node labeled by a symbol X, then

$$\mathcal{P}(\gamma) = \{X\}$$

- If the root of γ is labeled by X and has k children dominating subtrees $\gamma_1, \ldots, \gamma_k$ then

$$\mathcal{P}(\gamma) = \bigcup_{1 \leq i \leq k} X\mathcal{P}(\gamma_i)$$

For a tree set Γ .

$$\mathcal{P}(\Gamma) = \bigcup_{\gamma \in \Gamma} \mathcal{P}(\gamma)$$

2.1 Context-Free Grammars

Definition 2.1.1 A CFG, G, can be written as $G = (V_N, V_T, S, P)$ where

V_N is a finite set of nonterminal symbols

V_T is a finite set of terminal symbols

V_N and V_T are disjoint sets

S, the start symbol, belongs to V_N and

P is a finite set of productions of the following form:

$$A \rightarrow \alpha \qquad \alpha \in (V_N \cup V_T)^*, A \in V_N$$

The set of derivation trees (phrase-structure trees) produced by a CFG, is called a **local set.** From Thatcher's work [79, 80], it is obvious that the path set of every local set is a regular language[1]. Since this property of local sets is important in the context of this chapter, we will show how to construct a nondeterministic finite state machine that recognizes the path set of a given CFG.

Suppose we have a CFG, $G = (V_T, V_N, S, P)$ with tree set $T(G)$. Let us define a NFA, $M = (V_T \cup V_N, Q, q_S, F, \delta)$ such that $L(M) = \mathcal{P}(T(G))$. Let us assume that G is in Chomsky Normal Form. Let the set of states $Q = \{ q_X \mid X \in V_T, \text{ or } X \in V_N \}$. Let the set of final states $F = \{ q_a \mid a \in V_T \}$. For each production $A \rightarrow BC \in P$ we let $q_B \in \delta(q_A, A)$ and $q_C \in \delta(q_A, A)$. For each production $A \rightarrow a \in P$ we let $q_a \in \delta(q_A, A)$. If $S \rightarrow \epsilon \in P$ then include q_S in F.

CFG's are so named because the choice during a derivation of which production is used to rewrite a nonterminal is entirely independent of the derivation context. This choice of rule depends only on the identity of the nonterminal (which comes from some finite set).

[1]Thatcher actually characterized recognizable sets: for the purposes of this discussion we do not distinguish them from local sets.

2.2 Head Grammars

Head Grammars (HG's), introduced by Pollard [62], is a grammar formalism that manipulates pairs of strings. Not only is concatenation of these pairs possible, but **head wrapping** can be used to split a string and wrap it around another string. In fact, there are many other operations over pairs of strings. Some, but not all, of these operations can be simulated by the concatenation and wrapping operations of HG's. For an example see Chapter 4. The productions of HG's are very similar to those of CFG's except that the operation used must be made explicit.

Pollard [62] discusses the linguistic relevance of HG's, and a number of mathematical properties of Head Languages (HL's) are given in [65]. HG's are weakly equivalent to a number of other grammatical formalisms. It has been shown in [87, 43] that HG's and TAG's are weakly equivalent. In [89] the linguistic implications of this result are discussed. In [82] HG's and TAG's were proved to be weakly equivalent to a linear version of Indexed Grammars (defined in Section 2.4). In Chapter 5, we show the equivalence of Combinatory Categorial Grammars with these systems (also discussed in [88]).

One of the differences between HG's and CFG's is the ability of HG's to provide an account of discontinuous constituents in natural languages. Suppose that we are interested in giving an analysis of a string in which, on the basis of certain criteria, we decompose the string into parts one of which is not a substring of the original string. For example, the phrase "easy problem to solve" should perhaps be broken down into the constituents: "easy to solve" and "problem". The constituent "easy to solve" does not form a continuous substring of the final phrase, and is therefore discontinuous. In HG's, unlike CFG's, it is possible for a nonterminal to derive a discontinuous constituent such as this since the elements of the pair of substrings derived by a nonterminal need be adjacent in the derived string. Work on HG's has formed part of the motivation for the development of a system called Head-Driven Phrase Structure Grammars [63].

The definition of HG's that we use here differs slightly from the original definition given in [62], and we believe it to be a more uniform system than the original[2]. In our definition, we use *split* strings, which are written a pair of strings.

Definition 2.2.1 A **HG**, G, can be written as $G = (V_N, V_T, S, P)$ where

V_N is a finite set of nonterminal symbols

V_T is a finite set of terminal symbols

V_N and V_T are disjoint sets

S, the start symbol, belongs to V_N and

P is a finite set of productions having one of the following forms:

$$A \rightarrow C_i(\alpha_1, \ldots, \alpha_n) \quad \text{where } 1 \leq i \leq n$$
$$A \rightarrow W(\alpha_1, \alpha_2)$$

where $A \in V_N$ and $\alpha_1, \ldots, \alpha_n \in V_N \cup (V_T^* \times V_T^*)$

The operations C_i are concatenation operations, and the operation W is a wrapping operation. The string language $L(G)$ generated by a HG, G is defined as follows.

$$L(G) = \{\, w_1 w_2 \mid S \overset{*}{\underset{G}{\Longrightarrow}} \langle w_1, w_2 \rangle \,\}$$

[2]Pollard's original definition used **headed** strings or strings which (unless empty) contained a distinguished terminal called the **head**. The formalism suffered from the problem that the composition operations were partial functions. The problem arises from the fact that Pollard allows use of the empty string, which has no head. Thus, the composition functions were undefined whenever the headed empty string was supposed to contribute its nonexistent head to the resulting headed string.

The **derives** relation is defined such that $A \overset{*}{\underset{G}{\Longrightarrow}} \langle w_1, w_2 \rangle$ if the following holds.

$$A \rightarrow f(\alpha_1, \ldots, \alpha_n) \in P$$

and for each $1 \leq i \leq n$, either $\alpha_i = \langle w_{i,1}, w_{i,2} \rangle$ or $\alpha_i \overset{*}{\underset{G}{\Longrightarrow}} \langle w_{i,1}, w_{i,2} \rangle$ and

$$f(\langle w_{1,1}, w_{1,2} \rangle, \ldots, \langle w_{n,1}, w_{n,2} \rangle) = \langle w_1, w_2 \rangle$$

The function f will be one of the following functions.

$$C_i(\langle w_{1,1}, w_{1,2} \rangle, \ldots, \langle w_{i,1}, w_{i,2} \rangle, \ldots, \langle w_{n,1}, w_{n,2} \rangle) = \langle w_{1,1}w_{1,2} \ldots w_{i,1}, w_{i,2} \ldots w_{n,1}w_{n,2} \rangle$$

$$W(\langle w_{1,1}, w_{1,2} \rangle, \langle w_{2,1}, w_{2,2} \rangle) = \langle w_{1,1}w_{2,1}, w_{2,2}w_{1,2} \rangle$$

Example 2.2.1 Consider a HG generating $\{a^n b^n c^n d^n \mid n \geq 0\}$.

$$S \rightarrow C_1(\langle \epsilon, \epsilon \rangle) \qquad S \rightarrow C_2(\langle a, \epsilon \rangle, T, \langle d, \epsilon \rangle) \qquad T \rightarrow W(S, \langle b, c \rangle)$$

We can represent derivations of HG's with trees that encode the use of productions in a very similar way to the derivation trees for CFG's. For example, Figure 2.1 shows a derivation tree of the above grammar for the string *aabbccdd*. Each internal node is annotated by the operation (concatenation or wrapping) used to combine the pairs of strings that are derived by the daughters of that node.

Notice that HG derivation trees differ from phrase-structure trees. We can not simply read the derived string off the frontier of the tree from left to right (unless only concatenation operations were used). The increased complexity of the string languages of HG's arises not from derivation trees with more complex structure, but from a more complex "yield" function. CFG's can be seen as a special case of HG's in which only concatenation operations are used. The HG derivation process

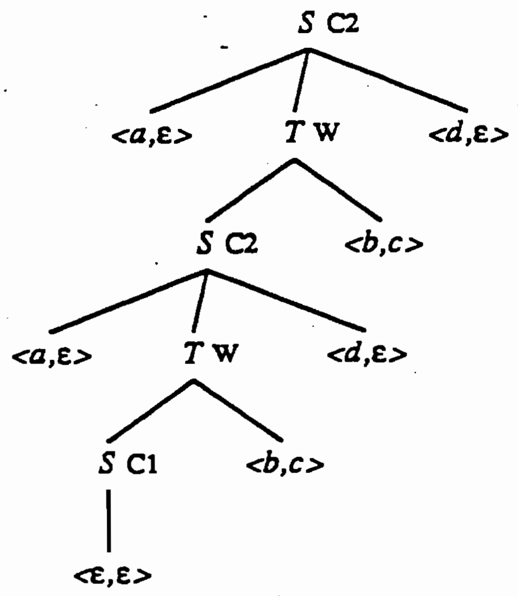


Figure 2.1: HG derivation tree

is essentially the same as that of CFG's. At each stage in the derivation a nonterminal is rewritten according to a production that matches that nonterminal in the grammar. In fact, derivation tree sets generated by HG's are similar to those of CFG's. Although there is an infinite number of operations (C_i for each $i > 0$), the number of operations in any grammar is finite. This annotation can be incorporated into the nonterminal labeling the nodes. Thus, the derivation trees of HG's are local sets, i.e., identical to those of CFG's.

2.3 Tree Adjoining Grammars

TAG's were first introduced in Joshi, Levy and Takahashi [42] and Joshi [36]. Certain changes have been made to the formalism since its inception (primarily concerning how local constraints are specified). The definition given here is derived from that given in Vijay-Shanker and Joshi [85], Joshi [38], and Vijay-Shanker [82]. TAG's differ from string rewriting systems such as CFG's in that they generate trees. These trees are generated from a finite set of so-called **elementary** trees using the operation of **tree adjunction**. There are two types of elementary trees: **initial** and **auxiliary**. Linguistically, initial trees correspond to nonrecursive phrase structure trees for basic sentential forms, whereas auxiliary trees correspond to modifying structures or complements. The use of these trees in a linguistically meaningful way has been discussed in a number of recent papers [51, 50, 48, 69, 40, 49, 52, 34]. The use of TAG's in generation is discussed in [39]. A number of mathematical results concerning the class of Tree Adjoining Languages (TAL's), (including certain closure properties, an upper bound for string recognition, and a string automaton) are given in [82]. A parsing algorithm based on the Earley algorithm for CFL's is discussed in [71, 70]. The complexity of their parallel recognition complexity is discussed in [58]. TAG's have been embedded in a unificational framework in [82, 83].

Definition 2.3.1 A **TAG** is a 5-tuple $G = (V_N, V_T, S, I, A)$ where

V_N is a finite set of nonterminals,

V_T is a finite set of terminals,

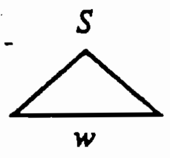
V_N and V_T are disjoint sets

S is a distinguished nonterminal,

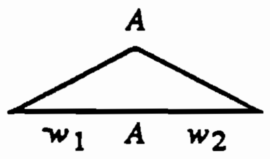
I is a finite set of initial trees and

A is a finite set of auxiliary trees.

Initial trees have the properties that their internal nodes are labeled by nonterminals; leaf nodes are labeled by either terminals or the empty string; and the root node is labeled by S.

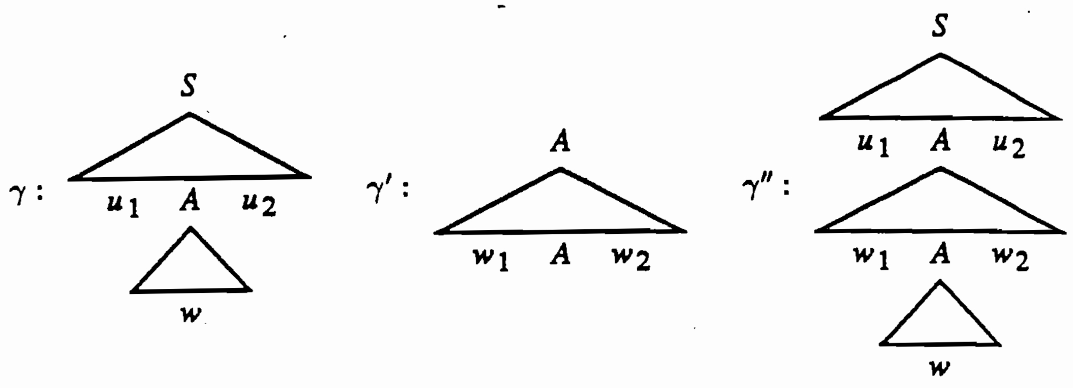


Auxiliary trees have the properties that their internal nodes are labeled by nonterminals; there is exactly one node on the frontier (the foot node) which is labeled by the same nonterminal that labels the root node; all other leaf nodes are labeled by either the empty string or a terminal.



The set $I \cup A$ is the set of **elementary trees**. There is one operation **adjunction** with which trees can be composed. Let η be some node labeled A in a tree γ at address a_η. Let γ' be a tree with root and foot labeled by A. When γ' is **adjoined** at η in the tree γ we obtain a tree γ''. The subtree under η is excised from γ, the tree γ' is inserted in its place and the excised subtree is inserted below the foot of

γ'. We say $\gamma'' = \gamma[a_\eta, \gamma']$.



We denote the adjunction of several trees $\gamma_1, \ldots, \gamma_k$ into a tree γ at distinct addresses $a_1, \ldots, a_k$, respectively, by

$$\gamma' = \gamma[a_1, \gamma_1] \ldots [a_k, \gamma_k]$$

The definition of adjunction allows more complex constraints to be placed on adjoining[3]. Associated with each node is a **selective adjoining** (SA) constraint specifying the subset of auxiliary trees which can be adjoined at this node. Trees can only be included in the SA constraint associated with a particular node if their root and foot are labeled with the same nonterminal that labels the node. A mechanism is provided for ensuring that adjunction is performed at a node. This is done by associating a **obligatory adjoining** (OA) constraint with that node[4].

If the SA constraint specifies an empty subset of trees, then adjunction cannot be performed at this node — we call this constraint the **null adjoining** (NA) constraint.

[3]In the original definition of TAG's constraints were associated with each tree. These constraints were used to determine the context surrounding a node at which the tree could be adjoined [41].

[4]In the feature structure-based TAG system [82, 83] these adjunction constraints are implicit in the feature structures and the success or failure of unification during the derivation.

Given a TAG, $G = (V_N, V_T, S, I, A)$, we define TAG derivations by induction on the number of steps in the derivation.

- $\mathcal{D}^0_G(\gamma) = \{\gamma\}$ for $\gamma \in I \cup A$ such that γ has no OA constraints.
- $\mathcal{D}^{h+1}_G(\gamma)$ is the set of all

  $$\gamma[a_1, \gamma'_1] \ldots [a_k, \gamma'_k]$$

  where $\gamma'_i \in \mathcal{D}^h_G(\gamma_i)$ for $1 \leq i \leq k$, and the auxiliary trees $\gamma_1, \ldots, \gamma_k \in A$ are included in the SA constraint at the k distinct nodes of γ with addresses $a_1, \ldots, a_k$, respectively. Also, the address of each OA node of γ must be included in $a_1, \ldots, a_k$.

$$\mathcal{D}_G(\gamma) = \bigcup_{i \geq 0} \mathcal{D}^i_G(\gamma)$$

If γ is an initial (auxiliary) tree then $\gamma' \in \mathcal{D}_G(\gamma)$ is referred to as a derived initial (auxiliary) tree. Note that in the definition of $\mathcal{D}_G$, derived auxiliary trees are adjoined into elementary trees, and all derived trees are complete, in the sense that they have no nodes with OA constraints.

The **tree set** of a TAG G is the set of trees

$$T(G) = \bigcup_{\alpha \in I} \mathcal{D}_G(\alpha)$$

The **string language** $L(G)$ of a TAG G is

$$L(G) = \{\, w \mid w \text{ is the frontier of some } \gamma \text{ in } T(G) \,\}$$

We now prove a theorem concerning the path sets of tree sets $T(G)$ where G is a TAG.

Theorem 2.3.1 For every tree set, $T(G)$ of a TAG, G, there exists a Context-Free language L such that $L = \mathcal{P}(T(G))$.

Proof:

We will define a pushdown automaton, M, for each TAG, G, such that $L(M) = \mathcal{P}(T(G))$, where M terminates on final state, i.e., the stack need not be empty. We give the construction, but omit the full proof of the result.

If $G = (V_N, V_T, S, I, A)$ then let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where

$$Q = \{q_0, q_1\} \qquad F = \{q_1\} \qquad \Sigma = V_N \cup V_T$$

$$\Gamma = \{Z_0, Z_1\} \cup \{\, \gamma \times n \mid \gamma \in I \cup A \text{ and } n \text{ is an address in } \gamma \,\}$$

We define δ as follows.

- For each initial tree $\alpha \in I$ let

$$(q_0, \langle \alpha, \epsilon \rangle\, Z_0) \in \delta(q_0, S, Z_0)$$

- If there is a node in a tree γ with address n, with no NA constraint, and β can be adjoined at n then

$$(q_0, \langle \beta, \epsilon \rangle\, \langle \gamma, n \rangle) \in \delta(q_0, \epsilon, \langle \gamma, n \rangle)$$

  We keep the node at which we adjoined on the stack so that if we follow a path through β from the root to foot, then we will be able to return to γ and continue from where we left off.

- If there is a node in a tree $\gamma \in I \cup A$ with address n, no OA constraint, and the i^{th} daughter of n, ni, is labeled $X \in V_N$, then if ni is an ancestor of the foot node of γ

$$(q_0, \langle \gamma, ni \rangle) \in \delta(q_0, X, \langle \gamma, n \rangle)$$

otherwise, (ni is not an ancestor of a foot node) let

$$(q_0, \langle\gamma, ni\rangle Z_1) \in \delta(q_0, X, \langle\gamma, n\rangle)$$

Z_1 is used to indicate that the contents of the stack below it are to be ignored. We do this because we are not following a path from the root to foot of an auxiliary tree. When we reach the frontier of γ we will have finished, we do not have to return to the tree that γ was adjoined at.

- If there is a foot node in a tree β with address n, then let

$$(q_0, \epsilon) \in \delta(q_0, \epsilon, \langle\beta, n\rangle)$$

We return to the tree into which β was adjoined.

- If there is a leaf node in a tree γ with address n, and labeled by $a \in V_T \cup \{\epsilon\}$ then let

$$(q_0, \epsilon) \in \delta(q_0, \epsilon, \langle\gamma, n\rangle)$$

We have finished and will have left either Z_0 or Z_1 on top of the stack.

- Finally, let

$$(q_1, \epsilon) \in \delta(q_0, \epsilon, Z_1)$$

$$(q_1, \epsilon) \in \delta(q_0, \epsilon, Z_0)$$

Notice that in both cases we have changed to state q_1 which indicates that we have finished.

$\square$

2.3.1 TAG Derivation Trees

Like the *strings* derived from a CFG or HG, the *trees* derived by a TAG do not encode their own derivation history. They do not show how the components of the grammar (in this case trees) were combined to produce the derived structure (the derived initial tree). It is possible to construct TAG's in which the same tree can be derived in several ways, e.g., by using different subsets of the elementary trees. This is a novel form of ambiguity that does not arise in the case of CFG's where each tree identifies a single derivation. The only aspect of a CFG derivation that is not encoded in a CFG derivation tree is the order in which nonterminals were rewritten. CFG derivation trees can be thought of representing a class of what are usually regarded as being entirely equivalent derivations.

In the case of CFG's, the strings are the **object level** structures, and the derivation trees the **meta level** structures. In comparing the tree sets of CFG's with TAG's, we should look at TAG derivation trees (meta level structures). They should indicate how intermediate (object level) trees were combined during the derivation to produce the derived tree. Derivation trees of this form were first defined in [82].

We define derivation trees inductively on the length of the derivation of a tree γ. We want the symbols that label nodes in derivations to refer to elementary trees. Therefore, we need to give explicit names to each of the elementary trees in the grammar. Let the set of elementary trees $I \cup A = \{ \gamma_1, \ldots, \gamma_m \}$ be given the names $\{ \overline{\gamma_1}, \ldots, \overline{\gamma_m} \}$.

- If γ_i is an elementary tree without OA constraints, the derivation tree consists of a single node labeled $\overline{\gamma_i}$ where $1 \leq i \leq m$.

- Consider the a derivation in which γ' is produced as follows.

$$\gamma' = \gamma_i[a_1, \gamma'_1] \ldots [a_k, \gamma'_k]$$

where $1 \leq i \leq m$ and $k \geq 1$, and all of the OA nodes in γ_i are included in $a_1, \ldots, a_k$.

For each $1 \leq j \leq k$, let γ'_j be derived from the elementary tree γ_{i_j}, where this derivation is represented by the derivation tree Υ_j. The root will be labeled by $\overline{\gamma_{i_j}}$.

Let Υ'_j be the tree Υ_j except that its root is relabeled as $\left\langle \overline{\gamma_{i_j}}, a_j \right\rangle$. The derivation tree for γ' will be rooted with a node labeled by $\overline{\gamma_i}$ with k subtrees, Υ'_j, for $1 \leq j \leq k$.

Example 2.3.1 Figure 2.2 shows a TAG derivation tree. It is best to read this tree from the bottom up. A tree was derived from the tree named $\overline{\beta_1}$ by adjoining two auxiliary trees named $\overline{\beta_2}$ and $\overline{\beta_3}$ into it addresses 12 and ϵ. This derived tree is adjoined into the initial tree named $\overline{\alpha}$ at address 1. The auxiliary tree $\overline{\beta_4}$ is also adjoined into $\overline{\alpha}$ at address 3.

From the definition of TAG's, it follows that the choice of which derived trees to adjoin into an elementary tree is not dependent on the history of the derivation, i.e., how the derived trees were produced. The choice is predetermined by a finite number of rules encapsulated implicitly in the SA constraints associated at nodes of elementary trees in the grammar. Thus, the *derivation trees* for TAG's have the same structure as local sets. For any TAG we can give a CFG whose tree set is exactly the set of derivation trees for the TAG. In Section 4 we propose one such scheme. As in the case of HG's, derivation structures are annotated; in the case of

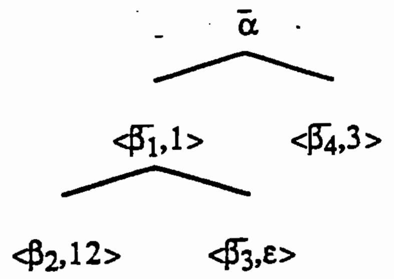


Figure 2.2: TAG derivation tree

TAG's, by the trees used for adjunction and addresses of nodes of the elementary tree where adjunctions occurred. In order to calculate the yield of a TAG derivation tree, we can not simply read the frontier of the tree from left to right. In fact, the terminals do not appear in the derivation tree, and the left to right ordering of nodes is not relevant in determining the yield of the tree[5].

In light of this discussion of TAG's, it is perhaps worth noting that CFG's can be interpreted as being notationally very similar to TAG's. We can think of a CFG as a tree manipulation system, where a grammar consists of a finite set of trees, that are combined with the tree substitution operation. Each of the trees in the grammar will have its root labeled by some nonterminal A having n children that are leaf nodes labeled by nonterminals or terminals $X_1, \ldots, X_n$. This tree directly corresponds to the production $A \rightarrow X_1 \ldots X_n$. Any tree rooted by A can be substituted for a *frontier* node in another tree that is also labeled by A. The string language of the CFG is the set of strings labeling the frontier of trees, all of whose leaf nodes are labeled by terminal symbols.

[5]Joshi (personal communication) has pointed out that there are cases (involving adjunction into root and nodes) where derivations that appear to be equivalent have distinct derivation trees. This suggests that the scheme given here may not be the best way of representing derivations.

Thus, a CFG can be associated with a set of object level trees, and a set of meta level (derivation) trees. Assuming that the derivation trees are analogous to those of TAG's, each node of the derivation tree could be labeled by the name of an elementary tree and tree address. If this scheme is used then the derivation tree will be essentially the same as the object level tree whose derivation it represents, except that all of the object level's terminal nodes will not be present in the derivation level tree. Thus, unlike TAG's, the family of derivation tree sets and object level tree sets of CFG's will be same.

2.3.2 Simple TAG's

In this section we consider how much hierarchical structure is needed in the elementary trees in order to get the full generative power of TAG's. This is a question that was raised in [37, 42], but the results in these papers do not apply to the current version of TAG's since the way in which adjunction constraints were defined differed from the present definition.

Definition 2.3.2 Define a **simple** tree as one that has at most two *nonterminals* on any path from the root to the frontier of the tree. A **simple TAG** (STAG) is a TAG in which every elementary tree is simple. Simple TAG string languages (STAL's) are the class of languages produced by STAG's.

Theorem 2.3.2 STAL = TAL

Proof: The inclusion of STAL in TAL is obvious since every STAG is a TAG. To show the other inclusion, we first give a construction from a arbitrary TAG to an equivalent STAG.

Let $G = (V_N, V_T, S, I, A)$ be an arbitrary TAG. We construct a STAG $G' = (\{ S \}, V_T, S, I', A')$. *With every nonterminal node of the elementary trees of G, there will be exactly one elementary tree in G'.* Note that S is the only nonterminal of G', so with each nonterminal node we need only specify the OA and SA constraints.

Consider some node η of some elementary tree γ. Let there be a simple tree γ_η, in G' whose root has an OA constraint if and only if η has an OA constraint. For each β in the SA constraints of η include the name of tree in G' corresponding to the root node of β in the SA constraints of the root of γ_η. Suppose η has k children $\eta_1, \ldots, \eta_k$ then the root of γ_η will have k children. $\eta'_1, \ldots, \eta'_k$. For $0 \leq i \leq k$, the node η'_i depends on η_i according to one of the following cases. Note that when $k = 0$ we have a foot node, and the tree in G' corresponding to this node will have just one node.

- If η_i is labeled by a terminal then η'_i is labeled by the same terminal.
- If η_i is the foot node of γ or η does not dominate a foot node and $i = 1$ then η'_i is the foot node of the tree and has an OA constraint and a SA constraint that includes just the tree in G' corresponding to the node η_i of γ.
- Otherwise, let η'_i have an OA constraint, with a SA constraint that includes just the tree in G' for η_i of γ. γ_η should include a node below η'_i labeled by the empty string.

Since the order of derivation makes no difference, let us assume that the first phase of the derivation of a tree in G' is the construction of derived trees corresponding to elementary trees of G (we call these derived elementary trees). Once this phase is complete, the remaining derivations in G and G' are essentially identical. We now justify the claim that the "derived elementary" trees of G' and the elementary trees of G are equivalent. The elementary trees of G are constructed one level at a time by trees in G'. There is exactly one tree in G' for every nonterminal node of G. OA constraints are used on all non-root nonterminal nodes of trees in G' to ensure that the entire elementary tree in G is produced. The constraints at a node η of a tree γ in G will match those in the "derived elementary" tree of the STAG since we associate the corresponding constraints with the root node of the tree in G' for η used to build up γ. The only difference between the elementary trees of G and the "derived elementary" trees of G' is the inclusion of additional nodes labeled by the empty string in the trees of G'.[6] These additional nodes have absolutely no effect on the strings that the grammar produces. $\square$

2.4 Indexed Grammars

Indexed Grammars (IG's) and a corresponding automata were introduced by Aho [3, 4] as a generalization of CFG's, their weak generative power falling between CFL's and CSL's. An algebraic characterization of IG's is given in [54], and a pumping lemma is given in [33]. There has been recent interest in the application of IG's to natural languages. Gazdar [24] considers a number of linguistic analyses which IG's

[6]If we allowed the use of substitution in addition to adjunction then we could construct the elementary trees of G precisely.

(but not CFG's) can make.

Definition 2.4.1 An **IG**, G, is denoted by $G = (V_N, V_T, V_S, S, P)$ where

V_N is a finite set of nonterminals,

V_T is a finite set of terminals,

V_S is a finite set of stack symbols,

V_N, V_T and V_S are disjoint sets

$S \in V_N$ is the start symbol, and

P is a finite set of productions, having the following form.

$$A[\cdot\cdot x] \rightarrow \alpha_1 \ldots \alpha_n$$

where $x \in V_S^*$, and for each $1 \leq i \leq n$ $\alpha_i = A[\cdot\cdot y]$, $\alpha_i = A[z]$, or $\alpha_i = w$ where $A \in V_N$, $w \in V_T^*$, and $y, z \in V_S^*$.

The notation for stacks uses $[\cdot\cdot l]$ to denote an arbitrary stack whose top symbol is l. We have allowed IG productions to be of a fairly general form. A fixed number of symbol can be "popped" from the left-hand side stack. Each of the nonterminals on the right-hand side can be given either fixed sized stacks, or the unbounded stack from the left-hand side onto which a fixed number of additional stack symbols may be "pushed".

The derives relation is defined as follows.

If $A[\cdot\cdot x] \rightarrow \alpha \in P$

$$\beta_1 A[yx]\beta_2 \underset{G}{\Longrightarrow} \beta_1 \alpha' \beta_2$$

where α' is the string α with y substituted for each occurrence of $\cdot\cdot$

The language, $L(G)$, generated by G is

$$\{\, w \mid w \in V_T^*, S[\,] \overset{*}{\underset{G}{\Longrightarrow}} w \,\}$$

An IG can be viewed as a CFG in which each nonterminal is associated with a stack. Each production can push or pop symbols on the stack as can be seen in the following example. The derivation of IG's can be represented with derivation trees whose nodes are labeled by nonterminals and stacks. Each internal node and its children would show the use of one of the productions, just as in the case of derivation trees for CFG's.

Example 2.4.1 An IG with the following productions will generate the string language $\{ a^n b^n \mid n \geq 0 \}$. Derivation trees will be of the form shown in Figure 2.3.

$$\begin{array}{ll} S[\cdot\cdot] \rightarrow S[\cdot\cdot i] & S[\cdot\cdot] \rightarrow A[\cdot\cdot]B[\cdot\cdot] \\ A[\cdot\cdot i] \rightarrow aA[\cdot\cdot] & B[\cdot\cdot i] \rightarrow B[\cdot\cdot]b \\ A[] \rightarrow \epsilon & B[] \rightarrow \epsilon \end{array}$$

Tree sets that are often associated with IG's do not include the stack annotations. These trees are true phrase-structure trees, whose yield can be read off the tree from left to right. However, if we wish to consider *derivation* trees then the stacks associated with occurrences of nonterminals must be included explicitly in the tree. If these are omitted then the resulting trees may be ambiguous, i.e., the same tree could be produced by two distinct derivations (using different productions)[7].

The class of string languages generated by IG's is considerably larger than that of TAG's, and this is related to certain differences in their tree sets. Trees derived by IG's exhibit a property that is not exhibited by the trees sets derived by TAG's or CFG's. Informally, two or more paths can be *dependent* on each other: for example,

[7]There have been a number of papers concerning a related, (though somewhat different) family of tree sets whose string languages are IG's. This family (produced by Context-Free Tree Grammars) was defined by Rounds [66], and later characterized by tree automata in [72, 32].

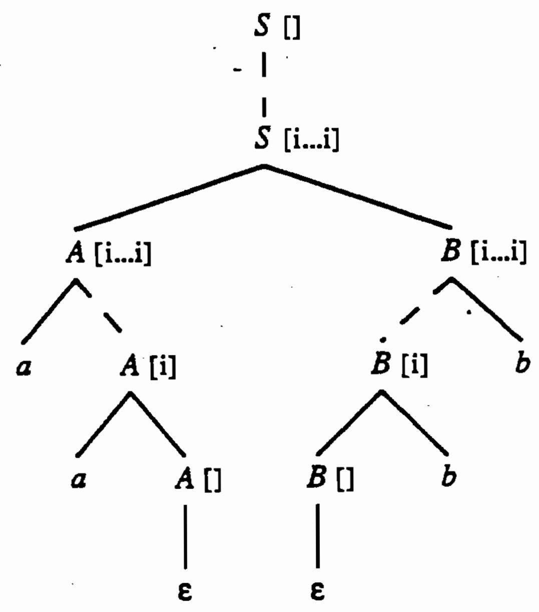


Figure 2.3: Dependent paths produced by a IG

they could be required to be of equal length, as in the trees of Figure 2.3. Although the path set for trees in Figure 2.3 is regular, no CFG or TAG generates such a tree set. IG's can produce tree sets in which trees have unboundedly large dependent subtrees with the use of productions in which an arbitrarily large stack is shared by more than one nonterminal on the right-hand side.

Gazdar [24] argues that sharing of stacks can be used to give analyses for coordination. Analogous to the sharing of stacks in IG's, Lexical-Functional Grammar's (LFG's) [44] use the unification of unbounded hierarchical structures. A number of other grammar formalisms can also do this [45, 61]. Unification is used in LFG's to produce structures having two dependent spines of unbounded length as in Figure 2.3. Bresnan, Kaplan, Peters, and Zaenen [12] argue that these structures are needed to describe crossed-serial dependencies in Dutch subordinate clauses. Providing a formalisms with this kind of power appears to lead to a dramatic increase in generative capacity. Berwick [9, 10] argues that it is just this sort of capability that leads to a grammar formalism's ability to generate "unnatural" languages, and that it should be avoided, since he believes that it is unnecessary.

2.5 Linear Indexed Grammars

Linear Indexed Grammars (LIG's) were considered by Gazdar [24], and are a restriction of Indexed Grammars introduced by Aho [3]. This system is called *Linear* Indexed Grammars because it can be viewed as a restriction of Indexed Grammars in which only one of the non-terminals on the right-hand side of a production can inherit the stack from the left-hand side, i.e., only one of the right-hand side non-terminals is associated with the unbounded stack denoting using "··". The other nonterminals on the right-hand side may be associated with any fixed sized stacks.

Example 2.5.1 The following productions form a LIG that generates the string language $\{ a^n b^n c^n d^n \mid n \geq 0 \}$

$$S[\cdot\cdot] \rightarrow aS[\cdot\cdot i]d \qquad S[\cdot\cdot] \rightarrow T[\cdot\cdot]$$
$$T[\cdot\cdot i] \rightarrow bT[\cdot\cdot]c \qquad T[\,] \rightarrow \epsilon$$

As a result of the restriction on the form of productions in LIG's, dependencies between unbounded branches are not possible in such a system. Vijay-Shanker [82] has shown that LIG's are weakly equivalent to TAG's (and HG's). In fact, the class of object level tree sets of TAG's are identical to the class of derivation tree sets of LIG's, if we omit the stacks associated with each node. In the case of the tree set of a LIG the nodes are labeled by arbitrarily large stacks. Thus, a LIG does not determine a bound on the number of node labels in its derivation trees.

2.6 Combinatory Categorial Grammars

In Chapter 5 we define CCG's and show the equivalence of Combinatory Categorial Grammars (CCG's) and LIG's. A corollary of one of the constructions used in this result is that the tree sets of CCG's are isomorphic to the tree sets of LIG's. Thus, all of the comments concerning the tree sets of LIG's also apply to the derivation tree sets of CCG's.

2.7 Multicomponent TAG's

An extension of the TAG system was introduced in [42] and later redefined in [38] in which the adjunction operation is defined on sequences of elementary trees rather than single trees. A principle underlying TAG's is that when we use TAG's to implement a linguistic theory we need only express relationships (dependencies) *locally.* The hypothesis is that dependencies need only be stated between nodes in the same tree; or between a node in an elementary tree and an entire tree that is adjoined at that node, i.e., a tree derived from a tree mentioned in the SA constraint. Thus, single tree forms the structure *within* which all of these linguistic dependencies must be stated. Joshi describes the elementary trees as the *domain of locality* over which dependencies can be stated. MCTAG's are an extension of TAG's that provide a mechanism to increase the domain of locality of TAG's by allowing us to group dependent symbols in one structure: a tree sequence. A MCTAG therefore consists of a finite set of finite elementary tree sequences. Since we only use one initial tree in any derivation, we assume that each initial tree sequence has length one. Grouping the elementary trees into sequences has two consequences. First, we must adjoin all trees in an auxiliary tree sequence at once in a single step in the derivation. Second, since a sequence of trees is now one object in the grammar, we have increase the domain of locality from a tree to a finite sequence of trees. This means that in a grammar we can express relationships between two nodes in different trees if both trees are in the same tree sequence. An example of such relationships would be the coindexing of two NP nodes or the equality of a feature of one node with a feature of another.

There are four ways of defining the adjunction operation with respect to tree sequence (multicomponent adjunction). We distinguish between elementary tree sequences and derived tree sequences. A derived tree sequence is a tree sequence obtained from an elementary tree sequence by any number of adjunctions into that sequence (including none).

1. All members of a sequence of trees are adjoined into distinct nodes of a *single elementary tree*, i.e., derivations always involve the adjunction of a derived auxiliary tree sequence into a single elementary tree.

2. Each member of a sequence of trees can be adjoined into distinct nodes of *any member of a single elementary tree sequence*, i.e, derivations always involve the adjunction of a derived auxiliary tree sequence into an elementary tree sequence.

3. Trees in a derived tree sequence are all adjoined into a *single derived tree*.

4. Each member of a derived sequence of trees can be adjoined into distinct nodes of *any member of a single derived tree sequence*.

The first of these definitions does not change either the weak generative capacity, or the tree sets that can be generated, compared with TAG's since OA constraints can be used to insure that either all or none of the members of a tree sequence are adjoined into the particular tree. The three other definitions all increase the generative capacity of the formalism and the extent of this increase appears to be related to the maximum size of auxiliary tree sequences. However, the third and fourth definitions allow the adjunction of trees into a set of nodes that may not have derived from an elementary object of the grammar. This violates certain intuitions related to the principal of locality. We therefore adopt the second of these definitions.

Definition 2.7.1 A **MCTAG** is a 5-tuple $G = (V_N, V_T, S, I, A)$ where

V_N is a finite set of nonterminals,

V_T is a finite set of terminals,

V_N and V_T are disjoint sets

S is a distinguished nonterminal,

I is a finite set of initial trees sequences of length 1, and

A is a finite set of auxiliary tree sequences.

The adjunction operation with respect to tree sequences (multicomponent adjunction) is according to the second alternative mentioned above, i.e., the trees in a a derived auxiliary tree sequence are adjoined into distinct nodes of trees in an elementary sequence. We extend the notation for adjunction to denote multicomponent adjunction as follows.

$$\begin{aligned}\langle\gamma_1,\ldots,\gamma_k\rangle\quad & [(i_{1,1},a_{1,1}),\ldots,(i_{1,m_1},a_{1,m_1})\left\langle\gamma'_{1,1},\ldots,\gamma'_{1,m_1}\right\rangle],\ldots,\\ & [(i_{n,1},a_{n,1}),\ldots,(i_{n,m_n},a_{n,m_n}),\left\langle\gamma'_{n,1},\ldots,\gamma'_{n,m_n}\right\rangle]\end{aligned}$$

That is, adjunction into $\langle\gamma_1,\ldots,\gamma_k\rangle$ occurs at the positions

$$(i_{1,1},a_{1,1}),\ldots,(i_{1,m_1},a_{1,m_1}),\ldots,(i_{n,1},a_{n,1}),\ldots,(i_{n,m_n},a_{n,m_n})$$

Each position $(i_{p,q}, a_{p,q})$ is a pair that identifies a node with address $a_{p,q}$ in the $i_{p,q}$th tree in the trees in $\langle \gamma_1, \ldots, \gamma_k \rangle$. For each sequence $\left\langle \gamma'_{q,1}, \ldots, \gamma'_{q,m_q} \right\rangle$ being adjoined, $(1 \leq q \leq n)$ there is a group of m_q positions preceding it, one for each tree in the sequence. The order of positions matches the order of trees in the sequence. Thus the pth tree in $\left\langle \gamma'_{q,1}, \ldots, \gamma'_{q,m_q} \right\rangle$ will be adjoined into $\gamma_{i_{q,p}}$ of $\langle \gamma_1, \ldots, \gamma_k \rangle$ at address $a_{q,p}$, for all $1 \leq q \leq n$, and $1 \leq p \leq m_q$.

Given a MCTAG, $G = (V_N, V_T, S, I, A)$, we define derivations formally.

- $\mathcal{D}^0_G(\langle \gamma_1, \ldots, \gamma_k \rangle) = \{\, \langle \gamma_1, \ldots, \gamma_k \rangle \,\}$

  for $\langle \gamma_1, \ldots, \gamma_k \rangle \in I \cup A$, where none of the trees $\gamma_1, \ldots, \gamma_k$ contain OA nodes.

- $\mathcal{D}^{h+1}_G(\langle \gamma_1, \ldots, \gamma_k \rangle)$ is the set of all

$$\begin{aligned} \langle \gamma_1, \ldots, \gamma_k \rangle \quad & [(i_{1,1}, a_{1,1}), \ldots, (i_{1,m_1}, a_{1,m_1}) \left\langle \gamma'_{1,1}, \ldots, \gamma'_{1,m_1} \right\rangle], \ldots, \\ & [(i_{n,1}, a_{n,1}), \ldots, (i_{n,m_n}, a_{n,m_n}), \left\langle \gamma'_{n,1}, \ldots, \gamma'_{n,m_n} \right\rangle] \end{aligned}$$

  such that $\left\langle \gamma'_{j,1}, \ldots, \gamma'_{j,m_1} \right\rangle \in \mathcal{D}^h_G(\langle \gamma_{j,1}, \ldots, \gamma_{j,m_1} \rangle)$, for $1 \leq j \leq n$, and according to the SA constraints at the nodes of trees in $\langle \gamma_1, \ldots, \gamma_k \rangle \in A$, the n auxiliary tree sequences

$$\langle \gamma_{1,1}, \ldots, \gamma_{1,m_1} \rangle, \ldots, \langle \gamma_{n,1}, \ldots, \gamma_{n,m_n} \rangle$$

  can be adjoined into $\langle \gamma_1, \ldots, \gamma_k \rangle$ at distinct positions

$$(i_{1,1}, a_{1,1}), \ldots, (i_{1,m_1}, a_{1,m_1}), \ldots, (i_{n,1}, a_{n,1}), \ldots, (i_{n,m_n}, a_{n,m_n})$$

  Also, adjunction must take place at all of the OA nodes in trees in $\langle \gamma_1, \ldots, \gamma_k \rangle$.

$$\mathcal{D}_G(\langle \gamma_1, \ldots, \gamma_k \rangle) = \bigcup_{i \geq 0} \mathcal{D}^i_G(\langle \gamma_1, \ldots, \gamma_k \rangle)$$

Note that in the definition of $\mathcal{D}_G$, derived auxiliary tree sequences are adjoined into elementary tree sequences. Since we wish to derived a single initial tree, we stipulate that initial tree sequences are always of length 1. Thus the definition of $\mathcal{T}(G)$ and $L(G)$ for MCTAG's is identical to that of TAG's.

MCTAG's are more powerful than TAG's, as can been seen by the following example.

Example 2.7.1 The MCTAG shown in Figure 2.4 generates the language $L_8 = \{ a_1^n \dots a_8^n \mid n \geq 0 \}$. The trees that are generated by this

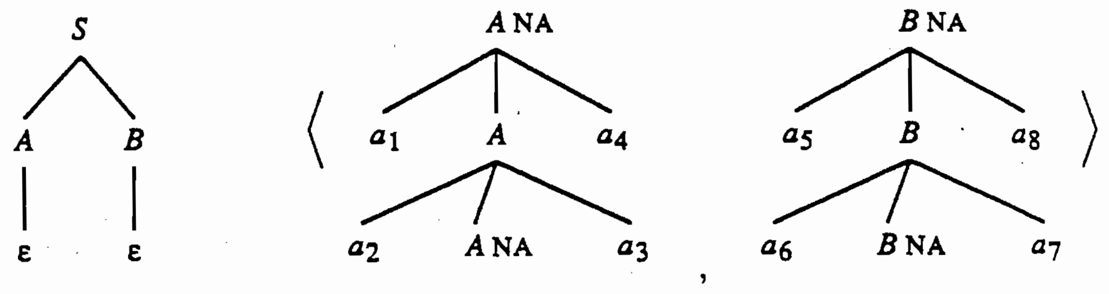


Figure 2.4: MCTAG for the language L_8

grammar will have the form of trees in Figures 2.5.

By increasing the number of trees in the auxiliary tree sequence, this example can be generalized to give grammars for any language $\{ a_1^n \dots a_k^n \mid n \geq 0 \}$ for some fixed k. In Chapter 4 we give a characterization of the weak generative power of MCTAG's.

The tree sets generated by MCTAG's differs from a TAG tree set in two ways. First, the path set may not be a CFL.

Example 2.7.2 The tree set of the MCTAG shown in Figure 2.6 has a

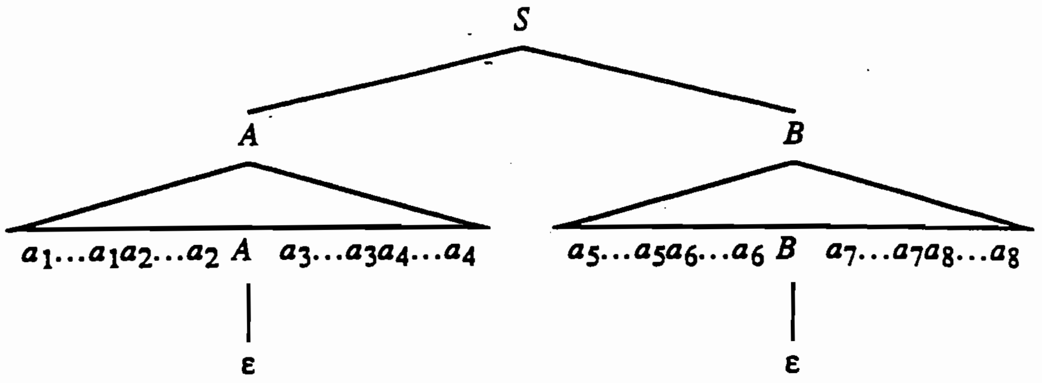

S
A
B
a1...a1a2...a2 A a3...a3a4...a4
a5...a5a6...a6 B a7...a7a8...a8
ε
ε

Figure 2.5: Dependent paths produced by an MCTAG

path set that is the language $L_3 = \{\, SA^nB^nC^n \mid n \geq 1 \,\}$.

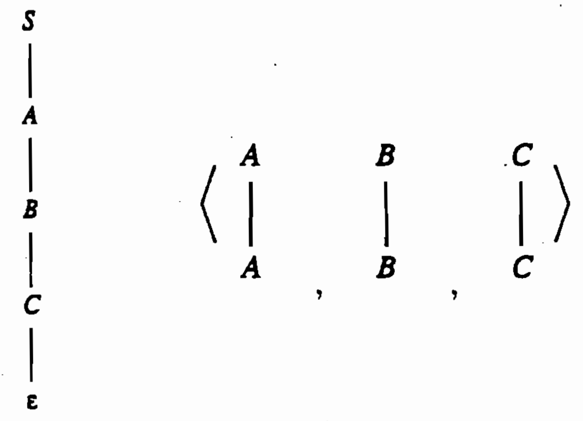


Figure 2.6: MCTAG whose path set is L_3

Second, the tree set has dependent paths as shown in Figure 2.5. In fact, MCTAG's can also generate tree sets of the form shown in Figure 2.3. The number of dependent paths in a MCTAG tree set is bounded by the length of the longest elementary tree sequence. This bound on the number of dependent paths leads to a difference between IG tree sets and MCTAG tree sets. The following example gives an IG whose tree set can not be generated by any MCTAG.

Example 2.7.3 An IG with the following productions will generate the string language $\{ a^{2^n} \mid n \geq 0 \}$. Derivation trees will be of the form shown in Figure 2.7.

$$S[\cdot\cdot] \rightarrow S[\cdot\cdot i] \qquad S[\cdot\cdot] \rightarrow A[\cdot\cdot]A[\cdot\cdot]$$

$$A[\cdot\cdot i] \rightarrow A[\cdot\cdot]A[\cdot\cdot] \qquad A[\,] \rightarrow a$$

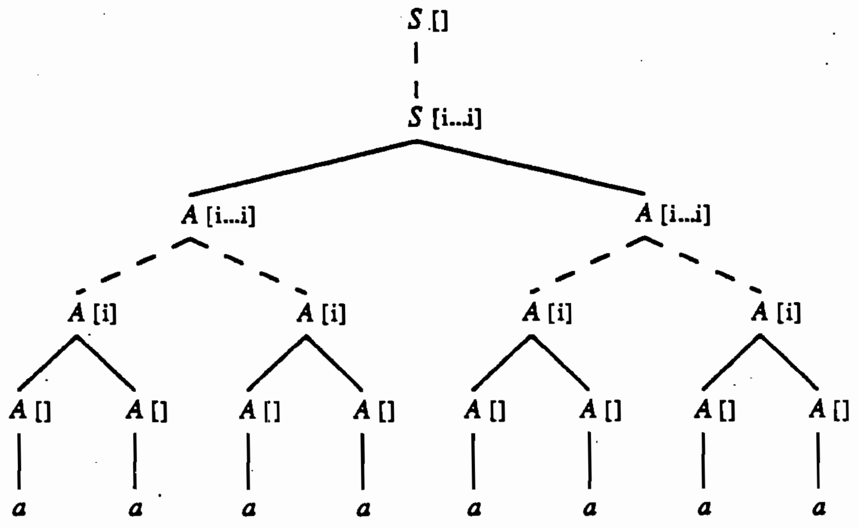


Figure 2.7: Unbounded number of dependent paths

In the trees shown in Figure 2.7 every path from root to frontier is identical and can therefore be said to be dependent. Thus the number of dependent paths is equal to the length of the derived string and thus, not bounded by the grammar.

2.7.1 MCTAG Derivation Trees

As in the case of TAG's, we must distinguish between object and meta level tree sets of MCTAG's. The encoding of MCTAG derivations as trees can be achieved by a similar method to that for TAG's. Instead of labeling nodes by the names of elementary trees and tree addresses, the nodes of MCTAG derivation trees are labeled by elementary tree sequences together with adjunction positions.

We define derivation trees inductively on the number of steps in the derivation of a tree sequence $\langle\gamma_1, \ldots, \gamma_k\rangle$. As in the case of TAG's, we must explicitly name the elementary trees with symbols that can be used in the labeling of nodes in the derivation trees. Let the set of all elementary tree sequences $I \cup A$ be enumerated as follows.

$$I \cup A = \{\, \langle\gamma_{1,1}, \ldots \gamma_{1,n_1}\rangle\,, \ldots, \langle\gamma_{m,1}, \ldots \gamma_{m,n_m}\rangle\, \}$$

Corresponding to this enumeration of the m tree elementary tree sequences, we give the tree sequences the names $\overline{\overline{\gamma_1}}, \ldots \overline{\overline{\gamma_m}}$, respectively.

- If $\langle\gamma_{i,1}, \ldots, \gamma_{i,n_i}\rangle$ is an elementary tree sequence with no OA nodes, where $1 \le i \le m$, the derivation tree consists of a single node labeled $\overline{\overline{\gamma_i}}$.
- Consider the a derivation in which $\langle\gamma'_1, \ldots, \gamma'_k\rangle$ is produced as follows.

$$\begin{aligned}\langle\gamma'_1, \ldots, \gamma'_k\rangle = \langle\gamma_{i,1}, \ldots, \gamma_{i,n_i}\rangle \quad & [(j_{1,1}, a_{1,1}), \ldots, (j_{1,l_1}, a_{1,l_1}), \left\langle\gamma'_{1,1}, \ldots, \gamma'_{1,l_1}\right\rangle], \\ \ldots, \quad & [(j_{n,1}, a_{n,1}), \ldots, (j_{n,l_n}, a_{n,l_n}), \left\langle\gamma'_{n,1}, \ldots, \gamma'_{n,l_n}\right\rangle]\end{aligned}$$

  where adjunction takes place into all of the OA nodes in $\langle\gamma_{i,1}, \ldots, \gamma_{i,n_i}\rangle$. For each $1 \le p \le n$, $\left\langle\gamma'_{p,1}, \ldots, \gamma'_{p,l_p}\right\rangle$ is derived from the elementary tree sequence with the name $\overline{\overline{\gamma_{i_p}}}$, where this derivation is represented by the derivation tree Υ_p whose root is labeled by $\overline{\overline{\gamma_{i_p}}}$. Let Υ'_p be the tree Υ_p except that its root is relabeled by $[(j_{p,1}, a_{p,1}), \ldots, (j_{p,l_1}, a_{p,l_p}), \overline{\overline{\gamma_{i_p}}}]$. The derivation tree for $\langle\gamma'_1, \ldots, \gamma'_k\rangle$ will be rooted with a node labeled by $\overline{\overline{\gamma_i}}$, and have n subtrees, Υ'_i for $1 \le p \le n$.

Notice that the number of node labels of MCTAG derivation trees is bounded by the grammar. Just as in the case of TAG's, the selective adjunction constraints of the MCTAG encapsulate a finite number of rules governing which adjunctions are permitted. As a result, like CFG's, TAG's, and HG's the *derivation* tree set of a MCTAG will be a local set. This reflects the fact that the choice of which multicomponent adjunctions should be made at a certain point in the derivation is independent of context. A MCTAG can be encoded as a finite set of rules each of which specifies that any tree sequence derived from a given elementary tree sequence can be adjoined at a given set of positions in another elementary tree sequence.

It is interesting to note that object level tree sets with dependent paths, and relatively complex path sets, can be generated by a formalism whose derivation trees are local sets. Thus, if a linguistic theory calls for a certain analysis that requires dependency between a bounded number of subtrees then it may turn out that MCTAG's have sufficient descriptive power for this. As we shall see in Chapter 4, MCTAG's can achieve this level of descriptive power without sacrificing certain desirable computational and linguistic properties associated with formalisms having reasonably constrained generative power.

2.8 Summary

In this chapter we have considered a variety of formalisms that are notationally very different and found that several of them share certain properties. CFG's, HG's, TAG's, and MCTAG's produce derivation trees that are structurally identical: they are local sets. In looking at derivation trees we abstract away from the formalism's notation, and can then compare formalisms that, because of their notational differences, have proved difficult to compare (other than in terms of their string languages). Our assumption is that properties of the derivation tree set reflect certain crucial computational and linguistic properties of the formalism.

A grammar formalism whose derivation trees sets are local sets has the property that a grammar can be expressed in the form of a finite set of "context-free like" rules, or productions. These rules may only be implicit in a traditional formulation of a grammar. The selection of which rules to apply at each stage of the derivation made independent of context. These context-free like rules can be thought of as rewrite rules, and the derivation process as context-free rewriting. Systems that have this property could therefore be called Context-Free Rewriting Systems.

As we have said, the derivation trees abstract away from the details of the structures that the formalisms manipulates, and the operations over these objects that are allowed. We must define a yield function for that formalisms which determines what structure has actually been derived in the derivation that is encoded by a derivation tree. In the case of CFG's, we simply concatenate the terminal symbols labeling the leaf nodes, from left to right. As we have seen however, things get progressively more complex with regard to HG's, TAG's, and MCTAG's. Calculating the "output" of a HG derivation tree involves the application of wrapping and concatenation operations on pairs of strings. For TAG's we must perform adjunction over trees, and for MCTAG's we perform multicomponent adjunction over tree sequences. Although wrapping, adjunction, and multicomponent adjunction are more complex operations than string concatenation, they all share certain properties with concatenation. In the Chapter 4 we will make some attempt to characterize the way in which these operations are restricted, however, this will be somewhat difficult since they each operate on different structures. Informally, each of these operations is "size preserving": i.e., the result of the operation is some structure that is formed by "adding" together its arguments in some way that they are not copied or erased.

They should also be "structure preserving", in the sense that they do not restructure their arguments. Chapter 4 investigates the class of formalisms whose derivation tree sets are local sets and whose composition operations are size preserving. We call these systems Linear Context-Free Rewriting Systems (LCFRS's).

In Chapter 3 we consider new formalisms that are also LCFRS's. These systems arise from a comparison of the differences between CFG's, TAG's and HG's.

Chapter 3

Progressions from CFL's to TAL's

3.1 Introduction

Four grammatical formalisms (TAG's, HG's, CCG's, and LIG's) have been shown to be weakly equivalent [87, 89, 82, 88], and Chapter 5. In this chapter we examine the relationship between this class of languages and CFL's. CFG's can generate the language $L_2 = \{ a^n b^n \mid n \geq 0 \}$ but not the language $L_3 = \{ a^n b^n c^n \mid n \geq 0 \}$. TAG's (and the other equivalent formalisms) can generate both of these languages as well as $L_4 = \{ a^n b^n c^n d^n \mid n \geq 0 \}$ but not the language $L_5 = \{ a^n b^n c^n d^n e^n \mid n \geq 0 \}$. Furthermore, CFG's, unlike TAG's, can not generate the copy language $L^2 = \{ ww \mid w \in V_T^* \}$; whereas neither CFG's or TAG's can generate the double copy language $L^3 = \{ www \mid w \in V_T^* \}$. When a formalism can generate the language $L_k = \{ a_1^n \ldots a_k^n \mid n \geq 0 \}$ it is said to be able to **count to** k. Thus, CFG's can count only up to 2 whereas TAG's can count up to at most 4. These observations suggest that the class of languages produced by these formalisms is in a certain sense a natural one, and that there is an interesting progression from CFL's to these slightly more powerful formalisms.

In this chapter we characterize this relationship between the class of languages generated by these formalisms and CFL's by showing several ways in which a natural progression can be established from CFL's to this language class. TAL's, HL's, CCL's and LIL's can be characterized in a variety of ways, several of which we make use of in this chapter. In particular, we will consider tree sets, grammars, automata, and generators as means of describing the differences between these formalisms and CFG's.

Grammars and Tree Sets In Chapter 2 we described several grammar formalisms and their associated tree sets. We found that CFG's produced tree sets that had regular path sets and independent paths, the path sets of TAG's had context-free path sets and independent paths. This suggests that there will be a family of tree sets with tree-adjoining language path sets and independent paths, and so on. It was also observed in Chapter 2 that CFG's and TAG's had similar derivation tree sets, and that the extra complexity of TAL's could be viewed as arising from a slightly more complex yield function being applied to derivation trees. In Section 3.2 we define a progression of TAG like formalisms. Their object level tree sets follow the progression suggested above, and hence their string languages also become increasingly complex. All of the members of the progression have derivation trees that are local sets, and hence the yield function changes as each stage in the progression. We also consider alternative grammatical characterizations of the same class of languages: HG's, and a novel kind of control grammars. For each of these formalisms we show how they can be generalized in a similar way producing progressions of language classes.

String Automata Another method used to characterize a class of string languages is in terms of the languages accepted by a family of automata. CFG's generate the same class of languages accepted by pushdown automata [14]. Vijay-Shanker [82, 84] describes an extension of the pushdown automaton that corresponds to TAL's. In Section 3.3 we compare these automata for CFL's and TAL's and define a progression of automata based on this comparison.

String Dependencies and Generators As we mentioned in Chapter 1, string dependencies has been used as a tool for characterizing the descriptive capacity of grammar formalism. The entire range of dependencies exhibited in a class of languages can be captured in a single language if that language generates the whole class when closed under certain simple operations. The Dyck language is such a language for CFL's and in Section 3.4 we describe the corresponding language for TAL's. The dependencies exhibited by CFL's and TAL's form the first two stages of a progression, that we describe in Section 3.4.

3.2 Progressions of Tree Sets and Grammars

3.2.1 Multi-level Tree Adjoining Grammars

In Chapter 2, it was observed that there are several formalisms whose derivation trees are structurally similar, reflecting common features of their derivation processes. Variations in the generative power of these systems results from differences in the way in which the labels of the derivation tree are interpreted. Each formalism has an associated "yield" function mapping derivation trees onto derived structures such as strings or trees. The yield of a derivation tree need not be the string on the frontier of the tree, in fact, it may not be a string at all. In the case of TAG's, the "yield" function outputs derived trees that resemble phrase-structure trees. TAG tree sets and local sets are in many ways very similar: they both have independent paths, and take their node labels from a finite set. As we saw in Chapter 2, the principal differences between them lies in their path sets. We also characterize families of tree sets that share the similarities of local sets and TAG tree sets, while generalizing the differences in their path sets.

Local sets have path sets that are regular languages, and string languages that are CFL's. TAG tree sets have path sets that are CFL's, and string languages that are TAL's. We will continue this progression, characterizing at the next stage tree sets with path sets that are TAL's, and string languages that fall in some class larger than TAL's, say $\mathcal{L}_3$. At the ith stage, we have tree sets with path sets in the class $\mathcal{L}_{i-1}$, and string languages in $\mathcal{L}_i$. The tree sets at every stage should have independent paths, and be labeled by members of a bounded sized alphabet.

The CFG yield function produces strings from phrase-structure trees; the TAG yield function produces CFG derivation trees (phrase-structure trees) from TAG derivation trees. In defining the string language of a TAG, the TAG and CFG yield function are composed. This is shown in Figure 3.1. The composition of the TAG

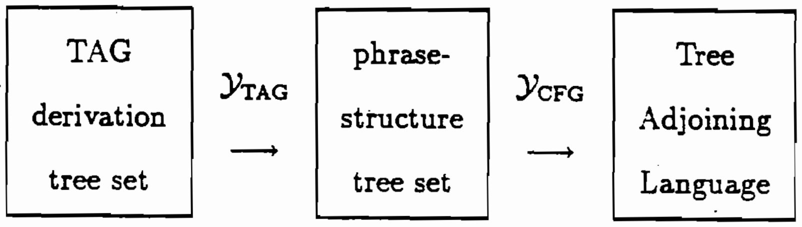

TAG derivation tree set
$\mathcal{Y}_{\mathrm{TAG}}$
phrase-structure tree set
$\mathcal{Y}_{\mathrm{CFG}}$
Tree Adjoining Language

Figure 3.1: Composition of CFG and TAG yield functions

and CFG yield function gives a composite yield function that when applied to local sets of TAG derivation trees, gives TAL's.

A natural generalization of this would involve producing more complex composite yield functions by composing yield functions. For example, consider a yield function whose *output* was TAG derivation trees. This yield function would then be of the appropriate type to be composed with the TAG yield function, followed by the CFG yield function. These three yield functions give a composite yield function that can be applied to appropriately labeled local sets to produce string languages. What form should these additional yield functions take?

As the TAG yield function has been described it maps from the TAG meta level to structures that are at both the TAG object level and the CFG meta level. This characterization is overly restrictive: the only constraint is that trees in the domain of the function be well formed derivation trees (meta-level trees) for some TAG. Consider a TAG G_1, whose object-level trees are meta-level trees of some other TAG, G_2. The elementary trees of G_1 are labeled by the names of trees in G_2, we will call these trees level-2. In this case, the yield functions of the two TAG's can be composed. This progression can be continued further, by considering level-3 trees: trees whose nodes are labeled by the names of trees whose nodes are labeled by the names of trees that are phrase-structure trees. Figure 3.2 illustrates the ith stage of this progression.

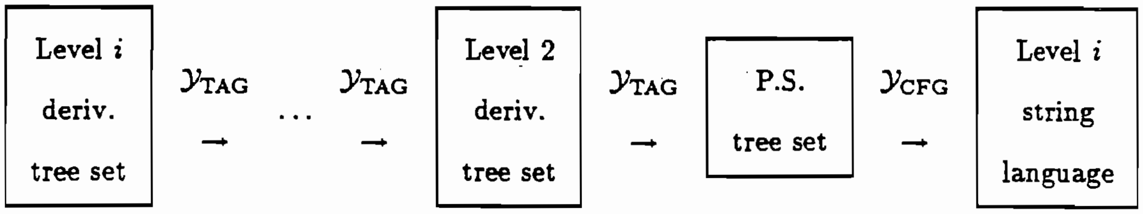

Level i deriv. tree set
$\mathcal{Y}_{\text{TAG}}$
...
$\mathcal{Y}_{\text{TAG}}$
Level 2 deriv. tree set
$\mathcal{Y}_{\text{TAG}}$
P.S. tree set
$\mathcal{Y}_{\text{CFG}}$
Level i string language

Figure 3.2: Composition of n yield functions

At each stage in the progression we include one additional TAG at level i with derivation (meta level) trees sets that are local sets, and whose object level tree sets form the TAG derivation trees or meta-level trees with respect to another TAG at the level below. The same TAG yield function is used at each stage to move down one level, until ultimately, we obtain phrase-structure trees to which the CFG yield function can be applied. The term Multi-level TAG's (MLTAG's) is used to refer to these formalisms. The composite yield function that applies to local sets of i-level trees will be the composition of $(i-1)$ TAG yield functions followed by the CFG yield function.

Let us define the TAG yield function. Let $G = (V_N, V_T, S, I, A)$ be a TAG such that $I \cup A = \{\gamma_1, \ldots, \gamma_k\}$. Remember that the elementary tree γ_i will be referred to in the derivation trees by $\overline{\gamma_i}$.

For a derivation tree Υ produced by the grammar G, the function $\mathcal{Y}_G$ is defined as follows.

- if Υ is a single node labeled by the tree named $\overline{\gamma_i}$ then $\mathcal{Y}_G(\Upsilon) = \gamma_i$ for $1 \leq i \leq k$.

- otherwise, if the label of the root of Υ is $\overline{\gamma_i}$, and $\Upsilon_1, \ldots, \Upsilon_n$ are the subtrees rooted at the addresses $1, \ldots, n$ in γ_i, respectively, then

$$\mathcal{Y}_G(\Upsilon) = \gamma_i[a_1, \mathcal{Y}_G(\Upsilon_1)], \ldots, [a_n, \mathcal{Y}_G(\Upsilon_n)]$$

  where $a_1, \ldots, a_n$ are the address in second component of the pair labeling the nodes with addresses $1, \ldots, n$ in Υ.

Let $\langle G_1, \ldots, G_n \rangle$ be a sequence of TAG's such that trees in the tree set of G_i are i-level trees, for $1 \leq i \leq n$. We define the object level tree set $T(G_1, \ldots, G_n)$ of a sequence $\langle G_1, \ldots, G_n \rangle$ as follows. For $n = 1$,

$$T(G_1) = \{\, \mathcal{Y}_{G_1}(\Upsilon) \mid \Upsilon \text{ is derivation tree of } G_1 \,\}$$

For $n > 1$

$$T(G_1, \ldots, G_n) = \{\, \mathcal{Y}_{G_1}(\Upsilon) \mid \Upsilon \in T(G_2, \ldots, G_n) \,\}$$

An alternative expression of this definition is as follows.

$$T(G_1, \ldots, G_n) = \mathcal{Y}_{G_1}(\mathcal{Y}_{G_2}(\cdots(\mathcal{Y}_{G_n}(D(G_n))\ldots)$$

where $D(G)$ is the set of derivation trees of G. From this definition, we see that every tree in $T(G_1, \ldots, G_n)$ is derived from some derivation tree of G_n by the composition of $\mathcal{Y}$ n times. As a final step we can apply the standard yield function (for phrase structure trees) to members of $T(G_1, \ldots, G_n)$ to obtain a string language. We have thus defined a generalization of TAG's, which we call multi-level TAG's. Although the trees in $T(G_1, \ldots, G_n)$ produced by the composite grammar $\langle G_1, \ldots, G_n \rangle$ become increasingly complex as n grows, the set of derivation trees for this grammar are exactly the set of derivation trees for G_n, which are therefore a local set.

3.2.2 Progression of Control Grammars

We turn to control grammars, an extension of CFG's in which derivations are "controlled". The notion of control grammars was previously discussed in [6, 21, 30]. For a general discussion of properties of control grammars see [68]. In this section we describe a variant of control grammars called Labeled Distinguished CFG's. We then define a generalization of these grammars that give a progression of language classes. We then describe an earlier attempt to describe a progression of control grammars producing classes of languages beyond CFL's.

Labeled Distinguished CFG's

In general a Control Grammar consists of a CFG whose productions are labeled. The general idea is that we can restrict (filter) the derivations of the grammar that are permitted. A set of **control words** indicating the productions used in the derivation is associated, in some way, with each derivation of a CFG. By specifying a set of such control words (**a control set**), we can define the language $L(G, C)$ generated by the grammar G, controlled by a control set C, as the set of strings in $L(G)$ having derivations whose control words are in C. We must define how control words are associated with a derivation. In our definition, a control word is associated with each path in the derivation, thus, at every point in a derivation, each symbol X_i in a sentential form $X_1 \ldots X_n$ is paired with a control word.

We wish to define a version of control grammars whose tree sets have independent paths. This requires a novel method of controlling derivations. In order to keep each path of the derivation independent we must not allow different paths to share control words. Whenever, according to a production, a nonterminal A is expanded by a string of symbols, only one of these symbols extends the control word associated with A; the other symbols begin recording the derivation from that point. This is done by specifying a symbol on the right-hand side of each production in the Labeled CFG's as distinguished (we show this by marking the chosen symbol with $\bar{}$). We shall refer to such CFG's as **Labeled, Distinguished CFG's** (LDCFG's).

Definition 3.2.1 A **LDCFG,** G, can be written $G = (V_N, V_T, V_L, S, P, L)$

where

V_N is a finite sets of nonterminals

V_T is a finite sets of terminals

V_L is a finite set of production labels

V_N, V_T and V_L are disjoint sets

$S \in V_N$ is the start symbol

P is a set of distinguished productions, and

L is a one-to-one function from P, to V_L.

Each member of P is a pair consisting of a standard Context-Free production and an integer corresponding to the position of one of the symbols on the right-hand side of the production. We will write the production $p = \langle \langle X, X_1 \ldots X_n \rangle, i \rangle$, where $L(p) = l$, and $0 \leq i \leq n$ as

$$p = l : X \rightarrow X_1 \ldots \check{X}_i \ldots X_n$$

where each $X_j \in V_n \cup V_T$, for $1 \leq j \leq n$, or if $n = 0$ then $i = 0$ and $X_0 = \epsilon$.

Sentential forms will associate a control word with each symbol (terminal and nonterminal), and will therefore be of the form $\langle X_1, w_1 \rangle \ldots \langle X_n, w_n \rangle$.

For a LDCFG $G = (V_N, V_T, V_L, S, P, L)$, we define relation $\underset{G}{\Longrightarrow}$ as follows.

$$\alpha_1 \langle X, w \rangle \alpha_2 \underset{G}{\Longrightarrow} \alpha_1 \langle X_1, \epsilon \rangle \ldots \langle X_i, wl \rangle \ldots \langle X_n, \epsilon \rangle \alpha_2$$

if P contains the production $p = l : X \rightarrow X_1 \ldots \check{X}_i \ldots X_n$.

The language $L(G, C)$, generated by $G = (V_N, V_T, V_L, S, P, L)$ and control set C is

$$\{ a_1 \ldots a_n \mid \langle S, \epsilon \rangle \overset{*}{\underset{G}{\Longrightarrow}} \langle a_1, w_1 \rangle \ldots \langle a_n, w_n \rangle, a_i \in V_T \cup \{ \epsilon \}, w_1, \ldots, w_n \in C \}$$

Since the control set is merely a language over an alphabet of labels of productions in a CFG, we can define this set as the language generated by another grammar. For example, the production sets of the control grammar for the language $\{ ww \mid w \in \{a, b\}^* \}$ is given below. Derivations involving the productions in P_1 are controlled by the language generated by a CFG whose productions are P_2.

Example 3.2.1

$$\begin{aligned} P_1 = \; & \{l_1 : S \rightarrow a\check{S}, & P_2 = \; & \{S' \rightarrow Tl_5, \\ & l_2 : S \rightarrow b\check{S}, & & T \rightarrow \epsilon, \\ & l_3 : S \rightarrow \check{S}a, & & T \rightarrow l_1Tl_3, \\ & l_4 : S \rightarrow \check{S}b, & & T \rightarrow l_2Tl_4\} \\ & l_5 : S \rightarrow \check{\epsilon}\} & & \end{aligned}$$

Higher-level Control Grammars

We can define a progression of Control Grammars. A grammar in the i^{th} level of the progression will consist of a LDCFG controlled by a language generated by a level $i-1$ control grammar.

Definition 3.2.2 $C^i = (G_1, \ldots, G_i)$ is an **ith order control grammar**, where

each $G_j = (V_{N,j}, V_{T,j}, V_{L,j}, S_j, P_j, L_j)$, $1 \leq j \leq i-1$, is a LDCFG;

$G_i = (V_{N,i}, V_{T,i}, S_i, P_i)$ is a standard CFG; and

for each j, $2 \leq j \leq i$, $V_{T,j} = V_{L,j-1}$.

We denote the class of languages generated by grammars in the i^{th} level of the progression by $\mathcal{C}^i$.

The language, $L(C^i)$ generated by $C^i = (G_1, \ldots, G_i)$ is defined by induction as follows. $L(C^i) = L(G_1, C)$ where $C = L(C^{i-1})$ and $C^{i-1} = (G_2, \ldots, G_i)$.

In Section 3.3.2 we show the relationship between this progression and a progression of automata that we define. Certain properties of this progression, particularly the complexity of recognizing its languages have been investigated in [57, 73].

The Hierarchy of Khabbaz

The existence of a progression of language classes, having a geometrically increasing ability to count symbols, was proposed by Khabbaz [46, 47]. The first class in his hierarchy was CFL's. Given a class at one level $\mathcal{L}$, the next higher class was that class generated by a formalism in which a **labeled linear CFG** is controlled by language in $\mathcal{L}$. A labeled linear CFG G controlled by a language L generates the following language. A derivation of a string is included only if the concatenation of labels of productions used in the derivation (the **control word**) is in the control language L. Since we are always controlling linear CFG's it is apparent that the classes in his hierarchy are not closed under concatenation and therefore not full AFL's.[1] The second member of Khabbaz's hierarchy appears to be equivalent to linear TAG's which are known not to be closed under concatenation[2]. It seems probable that Khabbaz used labeled linear CFG rather than arbitrary CFG for the following reason. At each point in a derivation of a linear CFG there is at most one nonterminal to expand, i.e., the leftmost and rightmost derivations are the same. However this breaks down in the case of non-linear CFG's. In a control grammar we must define a control word for any derivation, i.e., the string of labels giving the productions used in that derivation. This is straightforward for labeled linear CFG, but for arbitrary CFG's there are many ways to associate the control word with a derivation. We have given one such association which corresponds to Khabbaz's definition in the case of linear CFG's.

[1]Full AFL's are families of languages, first described by Ginsburg and Greibach [28], that are closed under concatenation, union, Kleene-star, arbitrary homomorphism, inverse homomorphism and intersection with regular languages.

[2]A linear TAG is a grammar in which any node in an elementary tree has at most one child labeled with a nonterminal [42].

3.2.3 Progression of Head Grammars

In this section we very briefly present a suggestion of how HG's might be generalized illustrating the difference between CFG's and HG's to and produce a progression of language classes that we conjecture will correspond to the other progressions discussed in this chapter. We will not attempt here to prove this relationship.

The primary distinction between HG's and CFG's is that HG's have three composition operations over pairs of strings, whereas CFG's involve concatenation of strings. For the ith level of the progression, we will define generalizations of HG's that use operations on k-tuples, where $k = 2^i$ for some $i \geq 0$.

The first operation is a generalization of the wrapping operation to k tuples.

$$W^k(\langle u_1, \ldots, u_k \rangle, \langle v_1, \ldots, v_k \rangle) = \langle u_1 v_1 v_2 u_2 u_3 v_3 v_4 u_4 u_5 \ldots u_{k-1} v_{k-1} v_k u_4 u_k \rangle$$

As in the case of HG's, there will be a number of concatenation operations. For each $i \geq 1$.

$$\begin{aligned} & C_i^k(\langle w_{1,1}, \ldots, w_{1,k} \rangle, \ldots, \langle w_{n,1}, \ldots, w_{n,k} \rangle) \\ = \; & \langle w_{1,1} \ldots w_{1,k} \ldots w_{i,1}, \ldots, w_{i,k} w_{n,1} \ldots w_{n,k} \rangle \end{aligned}$$

Note that these operations degenerate to the CFG operations for $k = 1$ and HG operations for $k = 2$.

Grammars using these operations, will in every other respect resemble HG's, i.e., they will be further examples of Generalized Context-Free Grammars, as described in [62]. Thus, they can be classified as Linear Context-Free Rewriting Systems (see Chapter 4).

3.3 Progression of String Automata

Vijay-Shanker [82] describes a string automaton (an nested pushdown automaton) that recognizes exactly the class of Tree Adjoining Languages. We shall describe this automaton and relate it to the pushdown automaton (PDA) that characterizes CFL's. We then show how this automata can be generalized to produce an infinite progression of automata. It should be noted that the automata differs from the nested stack automaton [4] an automaton which correspond to IG's.

3.3.1 Nested Pushdown Automaton

A PDA consists of a pushdown store, a finite state control and a one-way read only input tape. A move of the PDA depends on the top stack symbol, the next input symbol, and the state. A move can read over the next input symbol, change state, and replaces the top symbol of the stack by a sequence of n ($n \geq 0$) stack symbols.

An nested pushdown automaton (NPDA) consists of a finite state control and a one-way read only input tape, together with a pushdown of nonempty pushdown stores. The move of the NPDA depends on the top symbol of the top pushdown, the next input symbol, and the state. A move can change state, read an input symbol and modify the pushdown of pushdowns. This modification has two phases. First, the top symbol A of the top pushdown store Υ is replaced by a sequence of n stack symbols $B_1 \ldots B_n$ ($n \geq 0$) yielding the store Υ'. In the second phase, Υ' is replaced by a sequence of k nonempty pushdown stores, including Υ' if it is nonempty. An example is shown in Figure 3.3. In this example, each new stack contains only one stack symbol. However, in general, any finite number of stack symbols can be placed on these stacks.

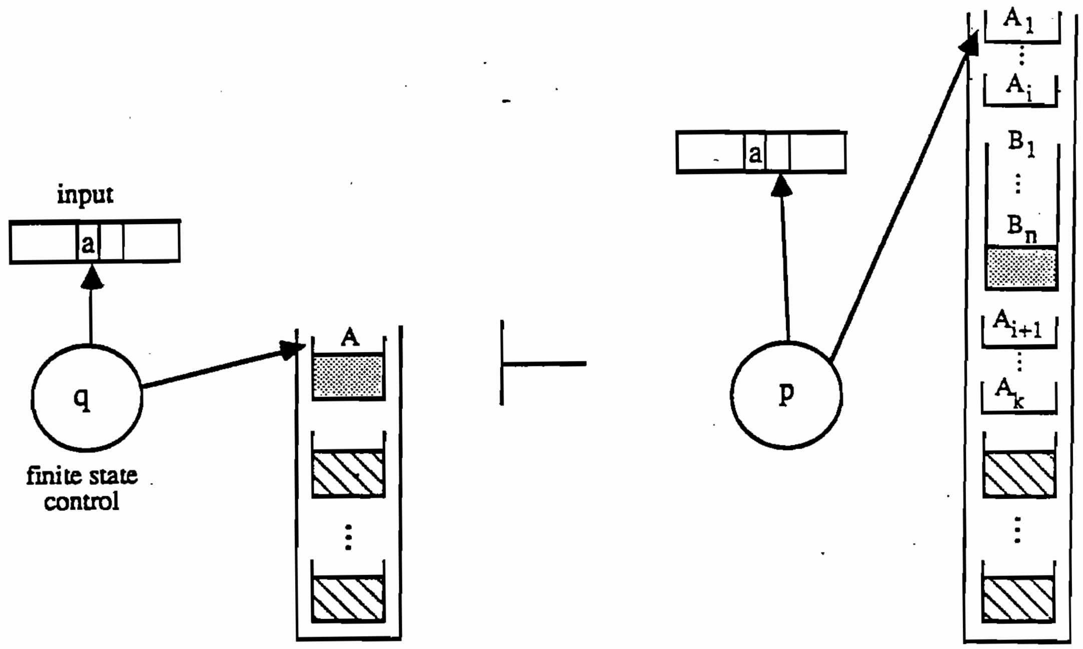


Figure 3.3: A move of an NPDA

The computation of an NPDA can be seen to correspond to the derivation of a TAG by relating the moves of an NPDA to the expansion of nodes in elementary trees in a top-down left-to-right order. When an NPDA imitates a TAG derivation, a stack is associated with each node. This stack encodes what is required to follow in the subtree under that node. Since we are concerned only with string recognition, a linearized encoding of the subtree can be stored in a stack. When we adjoin at a node we push an encoding of the spine of the auxiliary tree being adjoined on top of the stack. New stacks are placed above and below this stack encoding that part of the adjoined tree to the left and right of the spine respectively.

In several respects, this is a simple generalization of a PDA. Both the PDA and NPDA have a finite state control, and one-way read only input tape. Only the top symbol of the storage can be examined. The increase in power of the NPDA arising from the ability to insert symbols below an unbounded stack, something that can not be simulated by a PDA.

3.3.2 Progression of NPDA

If we consider the store of a PDA to be a 1st order data structure (pushdown list) consisting of stack symbols, the store of an NPDA will be a 2nd order pushdown consisting of 1st order stores. Thus, a progression can be established in which the ith order NPDA manipulates ith order stores. We define an ith order store as either a pushdown of stack symbols for $i = 1$ or as a pushdown of $(i-1)th$ order stores, for $i \geq 2$. An ith order NPDA consists of a finite state control, one-way read only input tape, and an ith order pushdown store. We will consider a $0th$ order store to be a single stack symbol. An automata with an 0th order store will be a finite state machine.

A transition of an ith order machine depends on the next input symbol, the state, and the top symbol on the store. The input can be read, the state changed, and the ith order store manipulated. The manipulations of an ith order store can be described recursively as a i step process: in terms of $(i-1)$ steps involving manipulation of an $(i-1)$th store, plus one additional step.

Consider some arbitrary ith order store, Υ_1^i. It will contain some finite number of $(i-1)$th order stores. Suppose, by induction, that Υ_1^{i-1}, the top of these $(i-1)$th order stores, can be transformed into Υ_2^{i-1} in $(i-1)$ stages. A transformation of Υ_1^i into Υ_2^i would involve removal of the top element Υ_1^{i-1} of Υ_1^i and its replacement by a finite number of $(i-1)$th order pushdown, one of which must be Υ_2^{i-1} if it is nonempty, the others being fixed sized stores. The basis of this recursive definition is given by the definitions of NPDA's.

Rather than showing moves in terms of a transition function, we shall express the legal moves of an automaton with a finite set of rewrite rules. Unbounded pushdowns are denoted by variables (lower case letters near the end of the alphabet with superscripts indicating the order of the pushdown). Stack symbols are specified in the rules with constants (capital letters near the beginning of the alphabet). Greek letters will be used to denote terms containing variables and constants. A pushdown is represented by a lists, and a nonempty ith order store can be written as follows: $[x^{i-1} \mid x^i]$. x^{i-1} is a variable that denotes the top $(i-1)$ level stack and x^i is a variable that denotes an arbitrary (possibly empty) list of the remaining $(i-1)$ order stores. For each $i \geq 1$ we define the class of ith order rules.

For $i = 1$ (PDA's) a first-order transition rules has one of the following forms,

$$\left\langle q, a, \left[A \mid x^1\right]\right\rangle \rightarrow \left\langle q', \left[A_1, \ldots, A_n \mid x^1\right],\right\rangle$$

$$\left\langle q, a, x^1\right\rangle \rightarrow \left\langle q', \left[A_1, \ldots, A_n \mid x^1\right],\right\rangle$$

where $q, q' \in Q$, $a \in \Sigma \cup \{\epsilon\}$, and $A, A_1, \ldots, A_n$ are stack symbols, $n \geq 0$.

Assume that α^i and β^i are such that

$$\left\langle q, a, \alpha^i\right\rangle \rightarrow \left\langle q', \beta^i\right\rangle$$

is an ith order move (note that α^i and β^i contain a mixture of constants and variables). An $(i+1)$th order move has the following form. If β^i is nonempty then

$$\left\langle q, a, \left[\alpha^i \mid x^{i+1}\right]\right\rangle \rightarrow \left\langle q', \left[s_1^i, \ldots, s_j^i, \beta^i, s_{j+1}^i, \ldots, s_n^i \mid x^{i+1}\right]\right\rangle$$

where $1 \leq j < n$, and $s_1^i, \ldots, s_n^i$ are constant ith order pushdowns. If β^i is empty then

$$\left\langle q, a, \left[\alpha^i \mid x^{i+1}\right]\right\rangle \rightarrow \left\langle q', \left[s_1^i, \ldots, s_n^i \mid x^{i+1}\right]\right\rangle$$

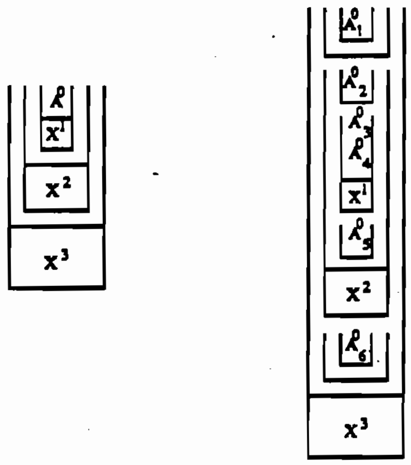
where $1 \leq j < n$, and $s_1^i, \ldots, s_n^i$ are constant ith order pushdowns. We also allow the following rule.

$$\left\langle q, a, x^{i+1} \right\rangle \rightarrow \left\langle q', \left[s_1^i, \ldots, s_n^i \mid x^{i+1}\right] \right\rangle$$

Example 3.3.1 We now give an example move of a 3rd order machine.

The state of the 3rd order pushdown before and after the move is also shown.

$$\left\langle q, a, \left[\left[\left[A \mid x^1\right] \mid x^2\right] \mid x^3\right] \right\rangle \rightarrow \left\langle q', \left[[[A_1]]\left[[A_2]\left[A_3 A_4 \mid x^1\right][A_5] \mid x^2\right][[A_6]] \mid x^3\right] \right\rangle$$



We write an **ith order NPDA** as $M = (Q, \Sigma, \Gamma, q_0, Z_0, F, R)$ where

Q is a finite set of states;

Σ is the input alphabet;

Γ is the stack alphabet;

$q_0 \in Q$ is the initial state;

$Z_0 \in \Gamma$ is the start symbol;

$F \subseteq Q$ is the set of final states; and

R is a finite set of ith order rewrite rules as described above.

The *yields* relation, $\vdash^*_M$, is defined directly from the rewrite rules. An instantaneous description of an ith order NPDA is a 3-tuple having the form $\langle q, w, s^i \rangle$ where: $q \in Q$; $w \in \Sigma^*$, the remainder of the input; and s^i is the current stack. The yields relation is a binary relation on instantaneous descriptions, and is defined as follows.

$$\left\langle q, aw, s^i \right\rangle \vdash_M \left\langle q', w, \psi(\alpha_2^i) \right\rangle \qquad \text{if} \qquad \left\langle q, a, \alpha_1^i \right\rangle \rightarrow \left\langle q', \alpha_2^i \right\rangle$$

where: ψ is a ground substitution such that $s^i = \psi(\alpha_1^i)$, $q \in Q$, and $a \in \Sigma \cup \{\epsilon\}$.

The language $L(M)$ accepted by an ith order machine $M = (Q, \Sigma, \Gamma, q_0, Z_0, F, R)$ is define thus.

$$L(M) = \{ w \mid \left\langle q_0, w, [^i Z_0]^i \right\rangle \vdash^*_M \left\langle q_f, \epsilon, [^i]^i \right\rangle, q_f \in F \}$$

We denote the class of all languages generated by NPDA in the ith level of the progression by $\mathcal{M}^i$.

The nested stack automaton [4] is a generalization of the stack automaton [29] in which stacks can be dynamically nested to arbitrary depth. The NPDA suggests a similar generalization of the PDA, in which the pushdowns can be nested dynamically to arbitrary depth during a computation. In defining the progression of NPDA in this section, we describe how pushdown of every nesting level can be manipulated. It would, therefore, be straightforward to modify this definition to give a machine that can increase the level of nesting dynamically. Since the level of nesting of the pushdown for each computation will always be finite, any language accepted by one of these machines will be acceptable by a member of the progression that has a level of nested as deep as needed. Thus, the languages accepted by this NPDA is presumably the union of the progression described here.

Example 3.3.2 Using the notation we have introduced, we describe a 3rd order NPDA M recognizing the language $L_8 = \{ a_1^n \dots a_8^n \mid n \geq 0 \}$.

$$\langle q_0, a_1, x^3 \rangle \rightarrow \langle q_1, [[[a_2][a_4]][[a_8]] \mid x^3]\rangle$$

$$\langle q_1, a_1, [[x^1 \mid x^2] \mid x^3]\rangle \rightarrow \langle q_1, [[[a_2 \mid x^1][a_4] \mid x^2][[a_8]] \mid x^3]\rangle$$

$$\langle q_1, \epsilon, x^3 \rangle \rightarrow \langle q_2, x^3 \rangle$$

$$\langle q_2, a_2, [[[a_2 \mid x^1] \mid x^2] \mid x^3]\rangle \rightarrow \langle q_2, [[x^1[a_3] \mid x^2][[a_7]] \mid x^3]\rangle$$

$$\langle q_2, \epsilon, x^3 \rangle \rightarrow \langle q_3, x^3 \rangle$$

$$\langle q_3, a_3, [[[a_3] \mid x^2] \mid x^3]\rangle \rightarrow \langle q_3, [x^2[[a_6]] \mid x^3]\rangle$$

$$\langle q_3, \epsilon, x^3 \rangle \rightarrow \langle q_4, x^3 \rangle$$

$$\langle q_4, a_4, [[[a_4] \mid x^2] \mid x^3]\rangle \rightarrow \langle q_4, [x^2[[a_5]] \mid x^3]\rangle$$

$$\langle q_4, \epsilon, x^3 \rangle \rightarrow \langle q_5, x^3 \rangle$$

$$\langle q_5, a_5, [[[a_5]] \mid x^3]\rangle \rightarrow \langle q_5, x^3 \rangle$$

$$\langle q_5, \epsilon, x^3 \rangle \rightarrow \langle q_6, x^3 \rangle$$

$$\langle q_6, a_6, [[[a_6]] \mid x^3]\rangle \rightarrow \langle q_6, x^3 \rangle$$

$$\langle q_6, \epsilon, x^3 \rangle \rightarrow \langle q_7, x^3 \rangle$$

$$\langle q_7, a_7, [[[a_7]] \mid x^3]\rangle \rightarrow \langle q_7, x^3 \rangle$$

$$\langle q_7, \epsilon, x^3 \rangle \rightarrow \langle q_8, x^3 \rangle$$

$$\langle q_8, a_8, [[[a_8]] \mid x^3]\rangle \rightarrow \langle q_8, x^3 \rangle$$

$$\langle q_8, \epsilon, [[[Z_0]] \mid x^3]\rangle \rightarrow \langle q_f, x^3 \rangle$$

The notation we use for ith order NPDA does not allow us to present, in a concise way, an ith order NPDA recognizing the language $L_{2^i} = \{ a_1^n \dots a_{2^i}^n \mid n \geq 0 \}$ for arbitrary i. We hope that is it is clear from the previous example that we could, in fact, define such an automaton.

Closure Properties of $\mathcal{M}^i$

In this section we show that $\mathcal{M}^i$ is a substitution-closed full AFL.

Theorem 3.3.1 $\mathcal{M}^i$ is closed under $\cap R$, h, h^{-1}, $\cup$, $\cdot$, kleene star, and substitution.

Proof: The proofs of each of these properties are analogous to those that have been given for the case of $\mathcal{M}^1$, or PDA's. We will very briefly describe each one.

$\cap R$ Suppose that for some k we have an instance of a k level machine, M^k. We have a finite state machine M accepting R. We produce a machine accepting $L(M^k) \cap L(M)$ using the cross product of the state sets of M and M^k as its state set. The start (resp. final) states are those containing start (resp. final) states of both M and M^k. The moves of the machine are restricted to those of M^k that are also legal in M with respect to its component of the states.

h This follows from closure under substitution.

h^{-1} Suppose that for some k we have an instance of a k level machine, M^k. Define a slight variant of M^k as follows. On input a, the machine stores the string $h^{-1}(a)$ in a finite buffer (part of the finite control) and M^k works on this strings as though it were the next symbols of the input. Once the buffer is empty the next real input symbols is considered and the process repeats. By finishing the string in the buffer before moving on to the next input symbol, the buffer can be of bounded size.

$\cup$ Suppose that we have two machines M_1^k and M_2^k with distinct state sets and stack symbols. A machine constructed from these that accepts the union of the languages, would nondeterministically choose to simulate one or other machine.

$\circ$ Suppose that we have two machines M_1^k and M_2^k with distinct state sets and stack symbols. A machine constructed from these that accepts the concatenation of the languages would begin by adding an special symbol on the bottom of the pushdown storage and then simulating M_1^k. There would be moves that on final states of M_1^k remove the special symbol from the bottom of the pushdown and move to the initial state of M_2^k.

Kleene Star This is similar to the previous case except that the two machines are the same and so it can repeat the computation of the machine any number of times.

Substitution Suppose that we have a machine M^k, and for each symbol a we have a machine M_a^k recognizing $L(a)$. We construct a machine that simulates M^k except that instead of reading an input symbol a it puts a marker on top of the pushdown and "calls" the machine M_a^k. Only when M_a^k has finished is the marker removed. In that way the pushdown of M^k will remain unaffected by the call to M_a^k.

□

At the end of the chapter we present a proof of the equivalence of the classes of languages produced by the control grammar progression in the classes of languages accepted by the NPDA progression, i.e., we prove the following theorem.

Theorem 3.3.2 $\mathcal{C}^i = \mathcal{M}^i \quad \text{for } i \geq 1$

3.4 Progression of Generators

We begin by describing the notion of a generator and explain how it can be used to characterize the string dependencies exhibited by a class of languages. We then describe the generator for TAL's and relate it to the generator for CFL's. In the final section we define a progression of generators based on the differences between the generators of CFL's and TAL's.

3.4.1 Generators

The following definition shows that a generator of a class of languages, has that property with respect to a specific set of operations on languages.

Definition 3.4.1 A **generator** for a family of languages $\mathcal{L}$ with respect to a certain set of operations O is a single language L, such that $\mathcal{L}$ is the smallest family of languages containing L that is closed under each of the operations in O.

What makes the generators that we discuss in this section useful for describing dependencies is the choice of the set of operations. The three operations that we use are intersection with regular languages, arbitrary homomorphism, and inverse homomorphism[3]. A homomorphism h, is a function from symbols in one alphabet to those in another. The homomorphism $h(w)$, of a string w is the string resulting from replacing the symbols of w by their image under h. The homomorphism $h(L)$,

[3]A class of languages that is closed under these these operations is called a **trio** [27].

of a language L, is the set of strings w' such that $w \in L$ and $h(w) = w'$. Given some homomorphism h, the inverse homomorphism $h^{-1}(L)$, of a languages L, is set of strings w such that there is a $w' \in L$ and $h(w) = w'$.

Loosely speaking, these operations have the effect of introducing new languages, but not new, more complex kinds of dependencies. This claim is illustrated by the fact that the class of regular languages, which exhibit the simplest forms of dependencies, are closed under these operations. Thus, any form of string dependency that is exhibited by some language in the class must also be present in the generator.

Generator for TAL's

It is a well known result that the **Dyck language** is a generator of CFL's with respect to arbitrary homomorphism, inverse homomorphism, and intersection with regular languages [19]. In this section we describe the Dyck language and then give a TAG that produces a language that is a generator for TAL's. In proving this result, we use an alternative language, denoted D_2^2, which is defined in terms of the intersection of two Dyck languages. We prove that D_2^2 is a generator of TAL's and that $T_2 = D_2^2$.

Definition 3.4.2 The **Dyck language** over $\Sigma = \{a_1, a_1', \ldots, a_n, a_n'\}$ is generated by $G = (\{S\}, \Sigma, S, P)$, where P contains the following productions.

$$S \rightarrow SS \qquad S \rightarrow \epsilon \qquad S \rightarrow a_i S a_i' \qquad 1 \leq i \leq n$$

Strings in a Dyck language are strings of matching pairs of brackets (some a_i and a_i'). The form of dependencies exhibited by the Dyck language, and hence CFL's are called *nested* dependencies. Each occurrence of an a_i is dependent on an occurrence of an a'_i and these dependencies are always nested. If we view a'_i as the right inverse of a_i ($a_i a'_i = \epsilon$), the Dyck language consists of all strings that can be reduced to ϵ, the identity.

Consider the language generated by the TAG in figure 3.4. We shall refer to

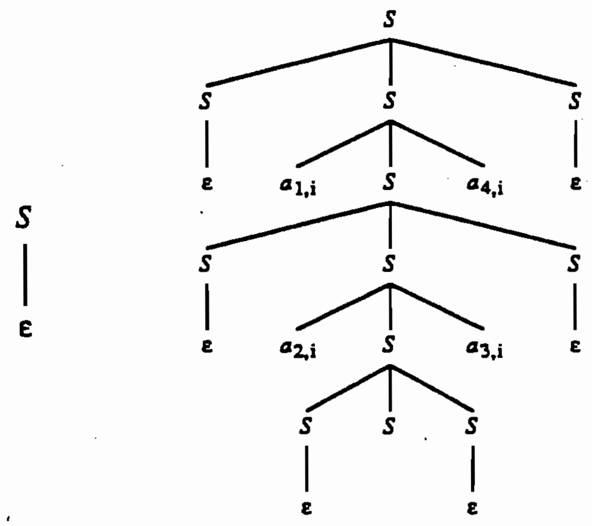


Figure 3.4: TAG for Tree Adjoining Dyck Language

the language generated by a TAG of this form with n sets of symbols as the tree adjoining dyck language (TADL) over n symbols, or simply T_n. We have proved the following theorem for TAL's which is analogous to the Chomsky-Schützenberger theorem for CFL's [19].

Theorem 3.4.1 For every TAL L, there is a positive integer n, a regular set R and a homomorphism h such that $L = h(R \cap T_n)$.

Proof: This theorem follows from Lemma 3.5.1 and 3.5.2 that we prove in Section 3.5. □

Notice that Theorem 3.4.1 involves an infinite number of languages T_n for each integer n. In fact, as shown in Lemma 3.4.1, there is a single language T_2 that when closed under inverse homomorphism produces the class of all T_n for all n. It is known [82] that TAL's are closed under intersection with regular languages, arbitrary homomorphism, and inverse homomorphism. Thus T_2 is the generator of TAL's with respect to these operations.

Lemma 3.4.1 There exists a homomorphism h such that $h^{-1}(T_2) = \bigcup_{n\geq 1} T_n$

Proof: We define h such that $h(a_{i,j}) = a_{i,1}a_{i,2}^{j}a_{i,1}$ for $1 \leq i \leq 4$ and $j \geq 1$. First, we outline the proof that $h^{-1}(T_2) \subseteq \bigcup_{n\geq 1} T_n$

Suppose $w \in h^{-1}(T_2)$. Then there is some $w' \in (T_2)$ such that $h(w) = w'$. Consider such a w'. Since it is in the range of h it must have the form $a_{i_1,1}a_{i_1,2}^{j_1}a_{i_1,1} \ldots a_{i_k,1}a_{i_k,2}^{j_k}a_{i_k,1}$ for some $i_1, \ldots, i_k$ and $j_1, \ldots, j_k$. Hence, $w = a_{i_1,j_1} \ldots a_{i_k,j_k}$. We can show by induction on the derivation of w' that if $w' \in T_2$ then $w \in T_n$, where n is the largest integer of $j_1, \ldots, j_k$.

We now describe how to show that $\bigcup_{n\geq 1} T_n \subseteq h^{-1}(T_2)$ Suppose $w \in \bigcup_{n\geq 1} T_n$. Let n be such that $w \in T_n$, and let $w = a_{i_1,j_1} \ldots a_{i_k,j_k}$. Let $h(w) = w'$. We can show by induction on the derivation of w in T_n that if $w \in T_n$ then $w' \in T_2$. $\square$

If we look at the form of dependent symbols in strings generated by T_n, we find that we produce strings with the following properties. Considering the six possible pairings between dependent $a_{1,i}$'s, $a_{2,i}$'s, $a_{3,i}$'s and $a_{4,i}$'s, we find that: $a_{1,i}$'s and $a_{2,i}$'s are nested; $a_{3,i}$'s and $a_{4,i}$'s are nested; $a_{1,i}$'s and $a_{4,i}$'s are nested; $a_{2,i}$'s and $a_{3,i}$'s are nested; $a_{1,i}$'s and $a_{3,i}$'s are crossed-serial; and $a_{2,i}$'s and $a_{4,i}$'s are crossed-serial. This suggests that there is a close relationship between T_n and Dyck languages.

The Dyck language (let us call it D) in which $a_{1,i}$'s are paired with $a_{2,i}$'s and $a_{3,i}$'s are paired with $a_{4,i}$'s generates all of the strings in T_n capturing the correct (nested) dependencies between $a_{1,i}$'s and $a_{2,i}$'s and between $a_{3,i}$'s and $a_{4,i}$'s. D is not equal to T_n because D also includes strings not in T_n in which dependent $a_{1,i}$'s and $a_{4,i}$'s or dependent $a_{2,i}$'s and $a_{3,i}$'s are *not* nested. However, consider a second Dyck language, D', in which we pair up each group of four symbols differently: $a_{1,i}$'s are paired with $a_{4,i}$'s; and $a_{2,i}$'s are paired with $a_{3,i}$'s. D', like D, includes all of the strings in T_n as well as strings not in T_n in which dependent $a_{1,i}$'s and $a_{2,i}$'s or dependent $a_{3,i}$'s and $a_{4,i}$'s are not nested. D' when intersected with D gives exactly T_n. By intersecting these two Dyck languages we have *simultaneously* imposed two different sets of nested dependencies: corresponding to the description given of dependencies in T_n.

With a single Dyck language, we were able to group dependent symbols in pairs. By intersecting two Dyck languages in this way we formed groups of 4 dependent symbols. Let us introduce some concepts that will help in making this relationship between the two generators more precise.

Definition 3.4.3 A **bracketing** B on an alphabet Σ is a binary relation such that: for every symbol a in Σ either $\langle a, a' \rangle \in B$ or $\langle a', a \rangle \in B$, $a \neq a', a' \in \Sigma$; and furthermore, for any symbol a in Σ a occurs in only one pair in the relation B.

For a bracketing relation, B, the Dyck language, $D(B)$, is generated by a CFG with the following productions.

$$S \rightarrow SS \qquad S \rightarrow \epsilon \qquad S \rightarrow aSa' \qquad \text{for each } \langle a, a' \rangle \in B$$

Let Σ_n be the finite alphabet

$$\Sigma_n = \{ a_{1,1}, a_{2,1}, a_{3,1}, a_{4,1}, \ldots, a_{1,n}, a_{2,n}, a_{3,n}, a_{4,n} \}$$

Define the two bracketings B_n and B'_n on the alphabet Σ_n as follows.

$$B_n = \{ (a_{1,i}, a_{2,i}), (a_{3,i}, a_{4,i}) \mid 1 \leq i \leq n \}$$

$$B'_n = \{ (a_{1,i}, a_{4,i}), (a_{2,i}, a_{3,i}) \mid 1 \leq i \leq n \}$$

Let $D^2_n = D(B_n) \cap D(B'_n)$.

In Section 3.5, we prove that $T_n = D^2_n$ and that for every TAL L there is a positive integer n, a regular set R and a homomorphism h such that $L = h(R \cap D^2_n)$. This serves to prove Theorem 3.4.1.

3.4.2 Progression of Generators

We define an infinite sequence of generators giving rise to successively larger classes of languages, the i^{th} generator involving the intersection of i Dyck languages and creating groups of 2^i dependent symbols. As in the case of the generator for TAL's, in each point in this progression, we decompose more complex types of "limited" crossing dependencies into the simultaneous use of several nested dependencies.

Notice that we have definitions of both CFL's and TAL's in terms of Dyck languages. This allows us to formalize the relationship between these two classes in terms of a progression of generators that are all defined using Dyck languages. We identify the progression between the generator for CFL's and the generator for

TAL's and then, by repeating this progression, produce a class of languages that has the same relationship to TAL's as TAL's have to CFL's.

Let Σ be the infinite alphabet $\Sigma = \{ a_1, a_2, \dots \}$. For each $n > 0$ let Σ_n be the first n members of this alphabet. B_n^k is a bracketing on Σ_{n2^k} defined as follows.

$$B_n^k = \{ (a_{(i-1)2^k+j}, a_{i2^k-j+1}) \mid 1 \leq i \leq n, 1 \leq j \leq 2^{k-1} \}$$

We define a sequence of generators as follows.

$$\text{For each } n \geq 1 \quad D_n^0 = \Sigma_n^*$$

For $i \geq 1$

$$\text{For each } n \geq 1 \quad D_n^i = D_{2n}^{i-1} \cap D(B_n^i)$$

The i^{th} family $\mathcal{L}^i$ is formed by closing D_2^i under the operations of intersection with regular languages, arbitrary homomorphism, and inverse homomorphism.

Theorem 3.4.2 $\mathcal{L}^0$ is the class of Regular languages.

Proof: One direction follows from the fact that the class of Regular languages is closed in intersection, homomorphism, and inverse homomorphism. Suppose R is a regular language over an alphabet Σ_1. Let h be a homomorphism mapping all elements of Σ_1 to a_1.

$$h^{-1}(D_2^0) = h^{-1}(\{ a_1, a_2 \}^*) = \Sigma_1^*$$

Thus R belongs to $\mathcal{L}^0$ since $R = \Sigma_1^* \cap R$ □

Theorem 3.4.3 $\mathcal{L}^1$ is the class of Context-free languages.

Proof: D_2^1 is the Dyck language on two pairs of letters. Therefore this is the Chomsky-Schützenberger Theorem [19]. □

We have shown that $\mathcal{L}^2$ is the class of TAL's, and therefore the class of HL's, CCL's, and LIL's.

Theorem 3.4.4 $\mathcal{L}^i$ is closed under GSM mappings and inverse GSM mappings, and substitution by regular sets.

Proof: These properties are all well known corollaries of the fact that each $\mathcal{L}^i$ is a trio (see [27, 35]). □

Let us define the class $\mathcal{L}$ as follows.

$$\mathcal{L} = \cup_{i \geq 1} \mathcal{L}^i$$

Theorem 3.4.5 Under the assumption that for all i $\mathcal{L}_i$ is properly included in $\mathcal{L}_{i+1}$ then $\mathcal{L}$ is not principal.

Proof: This can be shown by contradiction. Suppose L was a generator of $\mathcal{L}$. L must belong to $\mathcal{L}$, so for some k it belongs to some $\mathcal{L}^k$. But, $\mathcal{L}^k$ is an AFL so $\mathcal{L} \subseteq \mathcal{L}^k$ which contradicts the assumption that each class is properly included in the one above it. □

3.5 Proof of Characterization Theorem for TAL's

Lemma 3.5.1 $T_n = D_n^2$

Proof: In order to make the proof easier we use a slightly different grammar $G = (V_T, V_N, S, I, A)$, for T_n shown in Figure 3.5. Let G_1 and G_2 be the CFG's generating $D(\bar{B}_n)$ and $D(B'_n)$, respectively. To prove $T_n \subseteq D_n^2$ it is sufficient to show by induction that if $w_1 S w_2$ is the frontier of γ then

$$S \overset{*}{\underset{G_1}{\Longrightarrow}} w_1, \quad S \overset{*}{\underset{G_1}{\Longrightarrow}} w_2, \quad S \overset{*}{\underset{G_2}{\Longrightarrow}} w_1 S w_2$$

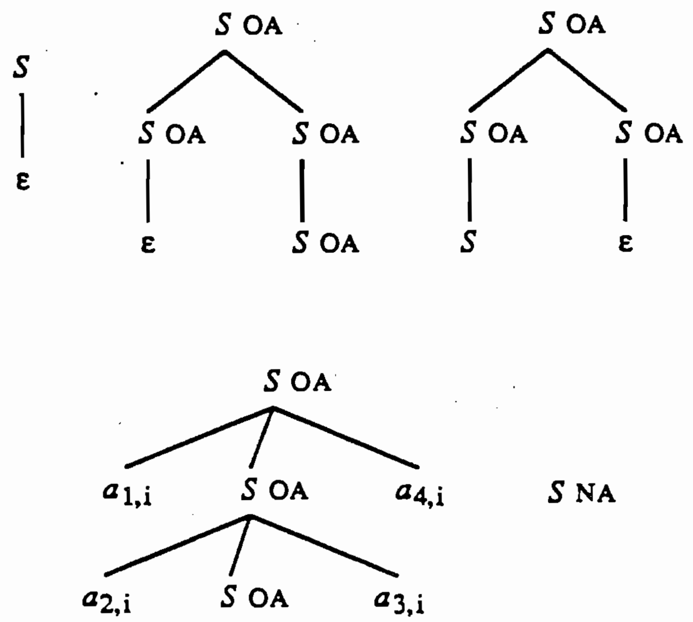
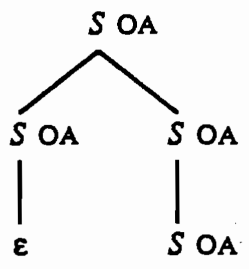


Figure 3.5: Alternative Tree Adjoining Dyck Grammar

For the basis of the induction consider each auxiliary tree in A with no OA nodes. It is straightforward to show that the required derivations in G_1 and G_2 are possible.

For the inductive case, consider a derived auxiliary tree $\gamma \in \mathcal{D}(\beta)$ derived in k steps.

- β is the tree:



Suppose $\gamma_1, \gamma_2, \gamma_3$ with frontiers $u_1Su_2, v_1Sv_2, w_1Sw_2$ where adjoined at the nodes with address $\epsilon, 1, 2$, respectively. By induction,

$$S \overset{*}{\underset{G_1}{\Longrightarrow}} u_1, \quad S \overset{*}{\underset{G_1}{\Longrightarrow}} u_2, \quad S \overset{*}{\underset{G_2}{\Longrightarrow}} u_1Su_2$$

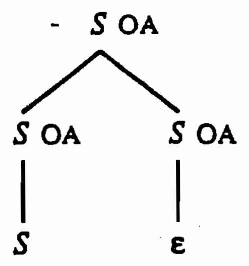
$$S \overset{*}{\underset{G_1}{\Longrightarrow}} v_1, \quad S \overset{*}{\underset{G_1}{\Longrightarrow}} v_2, \quad S \overset{*}{\underset{G_2}{\Longrightarrow}} v_1Sv_2$$

$$S \overset{*}{\underset{G_1}{\Longrightarrow}} w_1, \quad S \overset{*}{\underset{G_1}{\Longrightarrow}} w_2, \quad S \overset{*}{\underset{G_2}{\Longrightarrow}} w_1Sw_2$$

Thus we can obtain the following derivations.

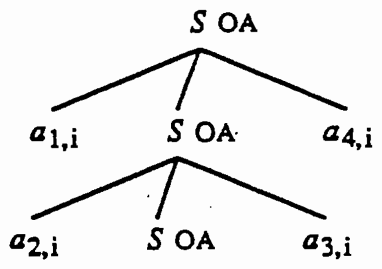
$$\begin{aligned}
S &\underset{G_1}{\Longrightarrow} SS \\
&\overset{*}{\underset{G_1}{\Longrightarrow}} u_1v_1 \\
S &\overset{4}{\underset{G_1}{\Longrightarrow}} SSSS \\
&\overset{*}{\underset{G_1}{\Longrightarrow}} v_2w_1w_2u_2 \\
S &\overset{*}{\underset{G_2}{\Longrightarrow}} u_1Su_2 \\
&\overset{*}{\underset{G_2}{\Longrightarrow}} u_1SSu_2 \\
&\overset{*}{\underset{G_2}{\Longrightarrow}} u_1v_1Sv_2w_1Sw_2u_2 \\
&\overset{*}{\underset{G_2}{\Longrightarrow}} u_1v_1Sv_2w_1w_2u_2
\end{aligned}$$

- β is the tree:



This case is very similar to the previous one.

- β is the tree:



Suppose $\gamma_1, \gamma_2, \gamma_3$ with frontiers $u_1Su_2, v_1Sv_2, w_1Sw_2$ where adjoined at the nodes with address $\epsilon, 2, 2 \cdot 2$, respectively. By induction,

$$S \overset{*}{\underset{G_1}{\Longrightarrow}} u_1, \quad S \overset{*}{\underset{G_1}{\Longrightarrow}} u_2, \quad S \overset{*}{\underset{G_2}{\Longrightarrow}} u_1Su_2$$

$$S \overset{*}{\underset{G_1}{\Longrightarrow}} v_1, \quad S \overset{*}{\underset{G_1}{\Longrightarrow}} v_2, \quad S \overset{*}{\underset{G_2}{\Longrightarrow}} v_1Sv_2$$

$$S \overset{*}{\underset{G_1}{\Longrightarrow}} w_1, \quad S \overset{*}{\underset{G_1}{\Longrightarrow}} w_2, \quad S \overset{*}{\underset{G_2}{\Longrightarrow}} w_1Sw_2$$

Thus we can obtain the following derivations.

$$\begin{aligned}
S &\overset{2}{\underset{G_1}{\Longrightarrow}} SSS \\
&\overset{*}{\underset{G_1}{\Longrightarrow}} u_1 S w_1 \\
&\underset{G_1}{\Longrightarrow} u_1 a_{1,i} S a_{2,i} w_1 \\
&\overset{*}{\underset{G_1}{\Longrightarrow}} u_1 a_{1,i} v_1 a_{2,i} w_1 \\
S &\overset{2}{\underset{G_1}{\Longrightarrow}} SSS \\
&\overset{*}{\underset{G_1}{\Longrightarrow}} w_2 S u_2 \\
&\underset{G_1}{\Longrightarrow} w_2 a_{3,i} S a_{4,i} u_2 \\
&\overset{*}{\underset{G_1}{\Longrightarrow}} w_2 a_{3,i} v_2 a_{4,i} u_2
\end{aligned}$$

$$\begin{aligned}
S &\overset{*}{\underset{G_2}{\Longrightarrow}} u_1 S u_2 \\
&\underset{G_2}{\Longrightarrow} u_1 a_{1,i} S a_{4,i} u_2 \\
&\overset{*}{\underset{G_2}{\Longrightarrow}} u_1 a_{1,i} v_1 S v_2 a_{4,i} u_2 \\
&\underset{G_2}{\Longrightarrow} u_1 a_{1,i} v_1 a_{2,i} S a_{3,i} v_2 a_{4,i} u_2 \\
&\overset{*}{\underset{G_2}{\Longrightarrow}} u_1 a_{1,i} v_1 a_{2,i} w_1 S w_2 a_{3,i} v_2 a_{4,i} u_2
\end{aligned}$$

To prove $D_n^2 \subseteq T_n$ we prove the following, by induction on the length of w. For this proof we use the original grammar for T_n given in Figure 3.4.

If $w \in D_n^2$ then there will be a derived tree γ of G, with frontier w such that for all u_1, u_2, u_3 where $w = u_1 u_2 u_3$ and

$$u_1 a_{1,i} a_{2,i} u_2 a_{3,i} a_{4,i} u_3 \in D_n^2$$

for some i, then there will be a node in γ labeled S dominating exactly the string u_2.

We will consider the order of TAG derivations in which we always adjoin into a derived initial tree. The basis of the induction follows immediately. Suppose that $w \in D_n^2$ and $|w| = k$ for $k > 0$, then $w = u_1 a_{1,k} a_{2,k} u_2 a_{3,k} a_{4,k} u_3$ for some k $(1 \leq k \leq n)$ where $u_1, u_2, u_3 \in D_n^2$.

By induction, there will be a tree γ with frontier $u_1 u_2 u_3$ such that for all ways of writing $u_1 u_2 u_3$ as $v_1 v_2 v_3$ where $v_1 a_{1,m} a_{2,m} v_2 a_{3,m} a_{4,m} v_3 \in D_n^2$ for some i, there is a node labeled S dominating v_2. In particular, there will be a node η in γ dominating u_2. Let γ' be the result of adjoining the tree containing the symbols $a_{1,i}, a_{2,i}, a_{3,i}, a_{4,i}$ at η. The tree γ' is shown in Figure 3.6. To show that γ' satisfies the above proposition for $w =$

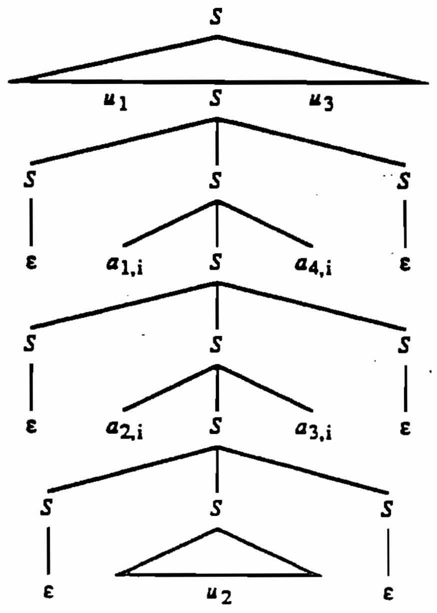


Figure 3.6: Derivation of Tree Adjoining Dyck Grammar

$u_1 a_{1,i} a_{2,i} u_2 a_{3,i} a_{4,i} u_3$ we must show that for all of the ways in which w can be split into strings w_1, w_2, and w_3 such that $w_1 a_{1,m} a_{2,m} w_2 a_{3,m} a_{4,m} w_3 \in D_n^2$ for some i, γ' contains a node dominating w_2. There are a large number of cases that must be considered to verify this, and we will not enumerate them here.

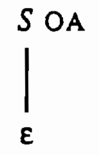
□

Lemma 3.5.2 For every TAL L there is a positive integer n, a regular set R and a homomorphism h such that $L = h(R \cap D_n^2)$.

Proof: Given a TAG, G, we give a construction of a right-linear CFG G', generating the regular language R. I will assume that the TAG has trees only of the form in which there is a single initial tree and 4 types of auxiliary trees.[4]

We choose a one-to-one correspondence, f, between the nodes in elementary trees and the first n integers where n is the total number of nodes in all elementary trees. In this way, we associate 4 symbols in Σ_n, $a_{1,f_{\gamma,\eta}}, \ldots, a_{4,f_{\gamma,\eta}}$ with each node, η of γ. We will use tree addresses [31] to denote nodes in the trees.

- Corresponding to single initial tree α

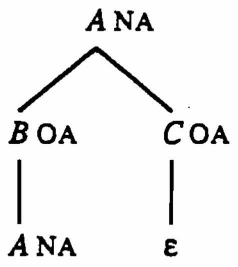


[4]This normal form is described in [87, 89, 43].

include the following productions.

$$S \to a_{1,f_{\alpha,0}} \overline{S}_l \qquad \underline{S}_l \to a_{2,f_{\alpha,0}} a_{3,f_{\alpha,0}} \underline{S}_r \qquad \overline{S}_r \to a_{4,f_{\alpha,0}}$$

- For each auxiliary tree β

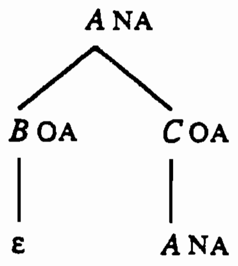


$$\overline{A}_l \to a_{1,f_{\beta,0}} a_{2,f_{\beta,0}} a_{1,f_{\beta,1}} \overline{B}_l \qquad \underline{B}_l \to a_{2,f_{\beta,1}} a_{1,f_{\beta,11}} a_{2,f_{\beta,11}} \underline{A}_l$$

$$\underline{A}_r \to a_{3,f_{\beta,11}} a_{4,f_{\beta,11}} a_{3,f_{\beta,1}} \underline{B}_r \qquad \overline{B}_r \to a_{4,f_{\beta,1}} a_{1,f_{\beta,2}} \overline{C}_l$$

$$\underline{C}_l \to a_{2,f_{\beta,2}} a_{3,f_{\beta,2}} \underline{C}_r \qquad \overline{C}_r \to a_{4,f_{\beta,2}} a_{3,f_{\beta,0}} a_{4,f_{\beta,0}} \overline{A}_r$$

- For each auxiliary tree β

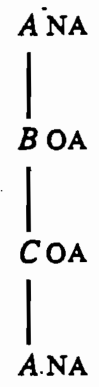


$$\overline{A}_l \to a_{1,f_{\beta,0}} a_{2,f_{\beta,0}} a_{1,f_{\beta,1}} \overline{B}_l \qquad \underline{B}_l \to a_{2,f_{\beta,1}} a_{3,f_{\beta,1}} \underline{B}_r$$

$$\overline{B}_r \to a_{4,f_{\beta,1}} a_{1,f_{\beta,2}} \overline{C}_l \qquad \underline{C}_l \to a_{2,f_{\beta,2}} a_{1,f_{\beta,21}} a_{2,f_{\beta,21}} \underline{A}_l$$

$$\underline{A}_r \to a_{3,f_{\beta,21}} a_{4,f_{\beta,21}} a_{3,f_{\beta,2}} \underline{C}_r \qquad \overline{C}_r \to a_{4,f_{\beta,2}} a_{3,f_{\beta,0}} a_{4,f_{\beta,0}} \overline{A}_r$$

- For each auxiliary tree β

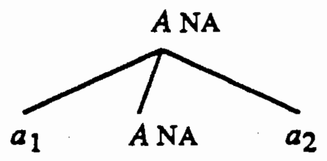


$$\overline{A}_l \rightarrow a_{1,f_{\beta,0}} a_{2,f_{\beta,0}} a_{1,f_{\beta,1}} \overline{B}_l \qquad \underline{B}_l \rightarrow a_{2,f_{\beta,1}} a_{1,f_{\beta,11}} \overline{C}_l$$

$$\underline{C}_l \rightarrow a_{2,f_{\beta,11}} a_{1,f_{\beta,111}} a_{2,f_{\beta,111}} \underline{A}_l \qquad \underline{A}_r \rightarrow a_{3,f_{\beta,111}} a_{4,f_{\beta,111}} a_{3,f_{\beta,11}} \underline{C}_r$$

$$\overline{C}_r \rightarrow a_{4,f_{\beta,11}} a_{3,f_{\beta,1}} \underline{B}_r \qquad \overline{B}_r \rightarrow a_{4,f_{\beta,1}} a_{3,f_{\beta,0}} a_{4,f_{\beta,0}} \overline{A}_r$$

- For each auxiliary tree β



$$\overline{A}_l \rightarrow a_{1,f_{\beta,0}} a_{2,f_{\beta,0}} a_{1,f_{\beta,1}} a_{2,f_{\beta,1}} a_{3,f_{\beta,1}} a_{4,f_{\beta,1}} a_{1,f_{\beta,2}} a_{2,f_{\beta,2}} \underline{A}_l$$

$$\underline{A}_r \rightarrow a_{3,f_{\beta,2}} a_{4,f_{\beta,2}} a_{1,f_{\beta,3}} a_{2,f_{\beta,3}} a_{3,f_{\beta,3}} a_{4,f_{\beta,3}} a_{3,f_{\beta,0}} a_{4,f_{\beta,0}} \overline{A}_r$$

The homomorphism h is defined such that $h(a_{i,f_{\gamma,j}}) = \epsilon$ for all i, j, and γ unless j is the address of a leaf node of γ labeled by a terminal a, in which case $h(a_{1,f_{\gamma,j}}) = a$.

We wish to show that $L(G) = h(D_n^2 \cap L(G'))$. Let G_1 be a CFG generating $D(B_n)$ and G_2 generate $D(B_n')$. We first show that $L(G) \subseteq h(D_n^2 \cap L(G'))$. We induct on the height of the derivation trees of trees derived in G. The basis of the induction simply involves considering each elementary tree in G and is straightforward. The inductive hypothesis is as follows.

Assume that for any tree $\gamma \in \mathcal{D}(\beta)$ for some $\beta \in A$ if the derivation tree for γ has height less that k then the following will all hold:

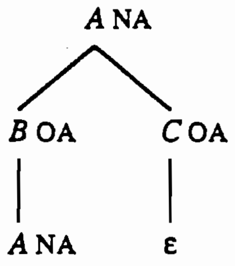
$$\overline{A}_l \overset{*}{\underset{G'}{\Longrightarrow}} w_1\underline{A}_l, \quad \underline{A}_r \overset{*}{\underset{G'}{\Longrightarrow}} w_2\overline{A}_r$$

where A labels the root of γ;

$$S \overset{*}{\underset{G_1}{\Longrightarrow}} w_1, \quad S \overset{*}{\underset{G_1}{\Longrightarrow}} w_2, \quad S \overset{*}{\underset{G_2}{\Longrightarrow}} w_1Sw_2 \quad h(w_1) = w_1', \quad h(w_2) = w_2'$$

where w_1' and w_2' form the frontier of γ to the left and right of the foot node.

There will be one case for each type of tree in A. All of the cases are very similar so we illustrate one of the cases. Suppose β is the tree following tree.



Two derived auxiliary trees γ_1 and γ_2 with frontiers u_1Bu_2 and v_1Cv_2 will have been adjoined into the nodes 1 and 2 of β, respectively. By induction the following holds.

$$\overline{B}_l \overset{*}{\underset{G'}{\Longrightarrow}} u_1\underline{B}_l, \quad \underline{B}_r \overset{*}{\underset{G'}{\Longrightarrow}} u_2\overline{B}_r$$

$$S \xRightarrow[G_1]{*} u_1, \quad S \xRightarrow[G_1]{*} u_2, \quad S \xRightarrow[G_2]{*} u_1 S u_2 \quad h(u_1) = u_1', \quad h(u_2) = u_2'$$

$$\overline{C}_l \xRightarrow[G']{*} v_1 \underline{C}_l, \quad \underline{C}_r \xRightarrow[G']{*} v_2 \overline{C}_r$$

$$S \xRightarrow[G_1]{*} v_1, \quad S \xRightarrow[G_1]{*} v_2, \quad S \xRightarrow[G_2]{*} v_1 S v_2 \quad h(v_1) = v_1', \quad h(v_2) = v_2'$$

By the construction, we have

$$\begin{array}{ll}
\overline{A}_l \to a_{1,f_{\beta,0}} a_{2,f_{\beta,0}} a_{1,f_{\beta,1}} \overline{B}_l & \underline{B}_l \to a_{2,f_{\beta,1}} a_{1,f_{\beta,11}} a_{2,f_{\beta,11}} \underline{A}_l \\
\underline{A}_r \to a_{3,f_{\beta,11}} a_{4,f_{\beta,11}} a_{3,f_{\beta,1}} \underline{B}_r & \overline{B}_r \to a_{4,f_{\beta,1}} a_{1,f_{\beta,2}} \overline{C}_l \\
\underline{C}_l \to a_{2,f_{\beta,2}} a_{3,f_{\beta,2}} \underline{C}_r & \overline{C}_r \to a_{4,f_{\beta,2}} a_{3,f_{\beta,0}} a_{4,f_{\beta,0}} \overline{A}_r
\end{array}$$

Thus we have the following derivations in G'.

$$\begin{array}{ll}
 & \overline{A}_l \\
\xRightarrow[G']{} & a_{1,f_{\beta,0}} a_{2,f_{\beta,0}} a_{1,f_{\beta,1}} \overline{B}_l \\
\xRightarrow[G']{*} & a_{1,f_{\beta,0}} a_{2,f_{\beta,0}} a_{1,f_{\beta,1}} u_1 \underline{B}_l \\
\xRightarrow[G']{} & a_{1,f_{\beta,0}} a_{2,f_{\beta,0}} a_{1,f_{\beta,1}} u_1 a_{2,f_{\beta,1}} a_{1,f_{\beta,11}} a_{2,f_{\beta,11}} \underline{A}_l \\
 & \underline{A}_r \\
\xRightarrow[G']{} & a_{3,f_{\beta,11}} a_{4,f_{\beta,11}} a_{3,f_{\beta,1}} \underline{B}_r \\
\xRightarrow[G']{*} & a_{3,f_{\beta,11}} a_{4,f_{\beta,11}} a_{3,f_{\beta,1}} u_2 \overline{B}_r \\
\xRightarrow[G']{} & a_{3,f_{\beta,11}} a_{4,f_{\beta,11}} a_{3,f_{\beta,1}} u_2 a_{4,f_{\beta,1}} a_{1,f_{\beta,2}} \overline{C}_l \\
\xRightarrow[G']{*} & a_{3,f_{\beta,11}} a_{4,f_{\beta,11}} a_{3,f_{\beta,1}} u_2 a_{4,f_{\beta,1}} a_{1,f_{\beta,2}} v_1 \underline{C}_l \\
\xRightarrow[G']{} & a_{3,f_{\beta,11}} a_{4,f_{\beta,11}} a_{3,f_{\beta,1}} u_2 a_{4,f_{\beta,1}} a_{1,f_{\beta,2}} v_1 a_{2,f_{\beta,2}} a_{3,f_{\beta,2}} \underline{C}_r \\
\xRightarrow[G']{*} & a_{3,f_{\beta,11}} a_{4,f_{\beta,11}} a_{3,f_{\beta,1}} u_2 a_{4,f_{\beta,1}} a_{1,f_{\beta,2}} v_1 a_{2,f_{\beta,2}} a_{3,f_{\beta,2}} v_2 \overline{C}_r \\
\xRightarrow[G']{} & a_{3,f_{\beta,11}} a_{4,f_{\beta,11}} a_{3,f_{\beta,1}} u_2 a_{4,f_{\beta,1}} a_{1,f_{\beta,2}} v_1 a_{2,f_{\beta,2}} a_{3,f_{\beta,2}} v_2 a_{4,f_{\beta,2}} a_{3,f_{\beta,0}} a_{4,f_{\beta,0}} \overline{A}_r
\end{array}$$

We can also show that the required derivations in G_1 and G_2 are possible.

We consider the other inclusion, i.e., $h(D_n^2 \cap L(G')) \subseteq L(G)$. We induct on the length of derivations in G'. It is clear that $S \xRightarrow[G']{*} w$ and $w \in D_n^2$

if and only if the following hold.

$$\overline{S}_l \overset{*}{\underset{G'}{\Longrightarrow}} w_1\underline{S}_l, \quad \underline{S}_r \overset{*}{\underset{G'}{\Longrightarrow}} w_2\overline{S}_r, \quad S \overset{*}{\underset{G_1}{\Longrightarrow}} w_1, \quad S \overset{*}{\underset{G_1}{\Longrightarrow}} w_2, \quad S \overset{*}{\underset{G_2}{\Longrightarrow}} w_1Sw_2$$

Thus, it is sufficient for us to prove the following, which we do by induction on the length of derivation in G'.

If the following hold

$$\overline{A}_l \overset{*}{\underset{G'}{\Longrightarrow}} w_1\underline{A}_l, \qquad \underline{A}_r \overset{*}{\underset{G'}{\Longrightarrow}} w_2\overline{A}_r$$

$$S \overset{*}{\underset{G_1}{\Longrightarrow}} w_1, \qquad S \overset{*}{\underset{G_1}{\Longrightarrow}} w_2, \qquad S \overset{*}{\underset{G_2}{\Longrightarrow}} w_1Sw_2$$

then there is a tree $\gamma \in \mathcal{D}(\beta)$ for some $\beta \in A$ such that the root of γ is labeled by A, $h(w_1) = w_1'$, and $h(w_2) = w_2'$ where w_1' and w_2' form the frontier of γ to the left and right of the foot node, respectively. It is clear that productions of G' must be used in groups corresponding to some elementary trees. There will therefore be one case in the induction for each group of productions. $\square$

3.6 Equivalence of Control Grammar and NPDA Progressions

Lemma 3.6.1 $\quad \mathcal{C}^i \subseteq \mathcal{M}^i \quad$ for $i \geq 1$

Proof: In order to simplify the proof, we will assume that the control grammars are in CNF (for a proof that this can be done, see [57]). This will not change the power of the machine. We will prove by induction that for any level i control grammar, there is an equivalent level i machine, using a single state. We only need one state because we assume that the pushdown machines have been extended to allow for a bounded number of symbols can be read off the top of the pushdown store.

The basis of the induction consider a level 1 grammar which is a CFG, and it is well known that there will exist an equivalent PDA of the appropriate form [14].

Consider an i level control grammar $C^i = (G_1, \ldots, G_i)$. The grammar $C^{i-1} = (G_2, \ldots, G_i)$ is an $(i-1)$ level grammar (whose strings are control words for G_1). By induction, there will be a machine M^{i-1} recognizing $L(C^{i-1})$. Define a machine M^i recognizing $L(C^i)$ as follows. For each rewrite rule

$$\left\langle q, \epsilon, s_1^{i-1} \right\rangle \rightarrow \left\langle q, s_2^{i-1} \right\rangle$$

of the machine M^{i-1}, and for each nonterminal A of G_1 we include the following rule in rule set of M^i.

$$\left\langle q, \epsilon, \left[\left[{}^{i-1}A \right]^{i-1}, s_1^{i-1} \mid x^i \right] \right\rangle \rightarrow \left\langle q, \left[\left[{}^{i-1}A \right]^{i-1}, s_2^{i-1} \mid x^i \right] \right\rangle$$

For each rewrite rule

$$\left\langle q, l, s_1^{i-1} \right\rangle \rightarrow \left\langle q, s_2^{i-1} \right\rangle$$

of the machine M^{i-1}, we add a rewrite rule to M^i based on the type of production in G_1 that has the label l.

If G_1 contains $l : A \rightarrow \check{B}C$ then include the following in M^i.

$$\left\langle q, \epsilon, \left[\left[{}^{i-1}A \right]^{i-1}, s_1^{i-1} \mid x^i \right] \right\rangle \rightarrow \left\langle q, \left[\left[{}^{i-1}B \right]^{i-1}, s_2^{i-1}, \left[{}^{i-1}C \right]^{i-1}, S^{i-1} \mid x^i \right] \right\rangle$$

where S^{i-1} is the starting pushdown state of M^{i-1}.

If G_1 contains $l : A \to B\check{C}$ then include the following in M^i.

$$\left\langle q, \epsilon, \left[\left[{}^{i-1}A\right]^{i-1}, s_1^{i-1} \mid x^i\right]\right\rangle \to \left\langle q, \left[\left[{}^{i-1}B\right]^{i-1}, S^{i-1}, \left[{}^{i-1}C\right]^{i-1}, s_2^{i-1} \mid x^i\right]\right\rangle$$

If G_1 contains $l : A \to \breve{a}$ then include the following in M^i.

$$\left\langle q, \epsilon, \left[\left[{}^{i-1}A\right]^{i-1}, s_1^{i-1} \mid x^i\right]\right\rangle \to \left\langle q, \left[\left[{}^{i-1}a\right]^{i-1}, s_2^{i-1} \mid x^i\right]\right\rangle$$

In addition to these rules we add for each terminal a of G_1, the following rule.

$$\left\langle q, a, \left[\left[{}^{i-1}a\right]^{i-1} \mid x^i\right]\right\rangle \to \left\langle q, x^i\right\rangle$$

The pushdown of the machine M^i when it begins a computation should be $\left[\left[{}^{i-1}S_1\right]^{i-1}, S^{i-1}\right]$ where S^{i-1} is the start pushdown of M^{i-1}.

We describe how M^i is simulating a derivation of C^i. We first describe an instantaneous description (ID) of a level i control grammar. This is a representation of some intermediate point in a leftmost derivation of the grammar.

- For $i = 1$, an ID α^1 is a pair $\langle w, \alpha\rangle$ where w is a string of terminals, α is a string of terminals and nonterminals, and $w\alpha$ is a sentential form corresponding to a leftmost derivation of the grammar. The ID for a completed derivation of w will be $\langle w, \epsilon\rangle$. The ID for a start derivation is $\langle \epsilon, S\rangle$ where S is the start symbol of the grammar.

- For a control grammar $C^{i+1} = (G_1, \ldots, G_{i+1})$, a level $(i+1)$ ID α^{i+1} will be a pair, $\langle w, \langle X_1, \alpha_1^i\rangle \ldots \langle X_n, \alpha_n^i\rangle\rangle$ where each X_j is a terminal or nonterminal of G_1, and each α_j^i is an i level ID, encoding the point

that some level i leftmost derivation of the grammar $(G_2, \ldots, G_{i+1})$ has reached in deriving the control word for that branch of the derivation. The ID for a completed derivation of a string w will be $\langle w, \epsilon \rangle$. The ID for a start derivation is $\langle \epsilon, \langle S_1, s \rangle \rangle$ where S is the start symbol of G_1 and s is the start derivation of $(G_2, \ldots, G_{i+1})$.

By induction, we describe how a level i pushdown can encode what remains of a level i ID.

- For $i = 1$, an ID $\langle w, \alpha \rangle$ where $\alpha = X_1 \ldots X_n$ is encoded by the pushdown $[X_1, \ldots, X_n]$, where each X_j is a terminal or nonterminal, $1 \leq j \leq n$.
- An $(i+1)$th level ID $\langle w, \alpha^{i+1} \rangle$ where $\alpha^{i+1} = \langle X_1, \alpha_1^i \rangle \ldots \langle X_n, \alpha_n^i \rangle$ is encoded by the $(i+1)$th order pushdown that pairs symbols with an encoding of what remains of the derivation of their control word.

  $$\left[\left[{}^{i-1}X_1\right]^{i-1}, s_1^i, \ldots, \left[{}^{i-1}X_n\right]^{i-1}, s_n^i\right]$$

  where s_j^i is the pushdown encoding α_j^i for $1 \leq j \leq n$. Since the control grammar is in CNF, in a legal derivation, when X_j is a terminal α_j^i must be ϵ and hence s_j^i will be empty and therefore not appear on the pushdown.

It can be shown that a transition is possible between two ID's of the grammar just in case a transition is possible in M^i between states encoding the two grammar ID's.

□

Lemma 3.6.2 $\mathcal{M}^i \subseteq \mathcal{C}^i$ for $i \geq 1$

Proof: Given an ith order NPDA $M^i = (Q, \Sigma, \Gamma, q_0, Z_0, F, R)$, we define an ith order control grammar $C^i = (G_1, \ldots, G_i)$, where for $1 \leq j < i$ $G_j = (V_{N,j}, V_{T,j}, V_{L,j}, Z_0, P_j, L_j)$ and $G_i = (V_{N,i}, V_{T,i}, Z_0, P_i)$, such that $L(M^i) = L(C^i)$. We assume that the NPDA terminates on final state and empty pushdown.

Before giving the construction, we will describe how top-down leftmost derivations of the grammars $G_1, \ldots, G_i$ will simulate the computation of M^i. The content of the pushdown will be encoded by the remaining unexpanded nonterminals in derivations of the grammars. The pushdown of M^i consists of $(i-1)$ order pushdowns. Intermediate sentential forms of G_1 encode the topmost symbol on each of these pushdowns. Intermediate sentential forms of the derivation of G_2 that is controlling the derivation of G_1 corresponding to the kth of the $(i-1)$ level pushdowns will encode the topmost symbol of each of the $(i-2)$ level pushdowns on this pushdown. Eventually, the sentential forms of G_i will encode the entire remaining contents of the k_{i-1}th pushdown of the k_{i-2}th 2nd order pushdown of the ... of the k_1th $(i-1)$ order pushdown, for each $k_1, \ldots, k_{i-1}$.

We will assume, without loss of generality, that i level pushdown transitions have the following form. For $i = 1$ the transitions have one of two forms.

$$\left[A \mid x^1\right] \vdash \left[A_1 A_2 \mid x^1\right] \qquad \text{or} \qquad \left[A \mid x^1\right] \vdash \left[x^1\right]$$

For $i > 1$, if $s_1^{i-1} \vdash s_2^{i-1}$ is an $(i-1)$ level transition, then the following are i level transitions.

$$\left[s_1^{i-1} \mid x^i\right] \vdash \left[\left[{}^{i-1}A_1\right]^{i-1} s_2^{i-1} \left[{}^{i-1}A_2\right]^{i-1} \mid x^i\right]$$

$$\left[s_1^{i-1} \mid x^i\right] \vdash \left[s_2^{i-1} \mid x^i\right]$$

where $A, A_1, A_2 \in \Gamma$. We say that $s_1^{i-1} \vdash s_2^{i-1}$ is the $(i-1)$ level component of this transition. We will make use of the inductive form of this definition, to decompose each level i pushdown transition into i components: one level k transition, for each $1 \leq k \leq i$.

We fix an enumeration of the rewrite rules in R so that each rule will be associated with a unique number j, where $1 \leq j \leq |R|$. We consider each rewrite rule and specify those productions that are to be included in the grammars of C^i, in order to encapsulate the rule.

Let the jth rule be of the form

$$\left\langle p, a, s_1^i \right\rangle \vdash \left\langle q, s_2^i \right\rangle$$

We will introduce productions depending on the i components of the transition $s_1^i \vdash s_2^i$. For each k where $1 < k < i$

- if the k level component is of the form

  $$\left[s_1^{k-1} \mid x^k\right] \vdash \left[\left[{}^{k-1}A_1\right]^{k-1} s_2^{k-1} \left[{}^{k-1}A_2\right]^{k-1} \mid x^k\right]$$

  then if the topmost symbol of s_2^{k-1} is a constant stack symbol $B \in \Gamma$ add the following production.

  $$l_{j,B}^k : A \rightarrow L_j^{k+1} A_1 \bar{B} A_2 \in P_{i-k+1}$$

where A is the topmost symbol of s_1^{k-1}. Alternatively, if s_2^{k-1} contains only variables then for all $X \in \Gamma \cup \{\epsilon\}$ include the production

$$l_{j,X}^k : A \rightarrow L_j^{k+1} A_1 \check{X} A_2 \in P_{i-k+1}$$

- if the k level component is of the form

$$\left[s_1^{k-1} \mid x^k\right] \vdash \left[s_2^{k-1} \mid x^k\right]$$

then if the topmost symbol of s_2^{k-1} is a constant stack symbol $B \in \Gamma$

$$l_{j,B}^k : A \rightarrow L_j^{k+1} \check{B} \in P_{i-k+1}$$

where A is the topmost symbol of s_1^{k-1}. Alternatively, if s_2^{k-1} contains only variables then for all $X \in \Gamma \cup \{\epsilon\}$ include the production

$$l_{j,X}^k : A \rightarrow L_j^{k+1} \check{X} \in P_{i-k+1}$$

We must deal with the two cases where $k = 1$ and $k = i$ First when $k = i$, if the i level component is of the form

$$\left[s_1^{i-1} \mid x^i\right] \vdash \left[\left[{}^{i-1}A_1\right]^{i-1} s_2^{i-1} \left[{}^{i-1}A_2\right]^{i-1} \mid x^i\right]$$

then if the topmost symbol of s_2^{i-1} is a constant stack symbol $B \in \Gamma$, then for each $r, s, t \in Q$, let

$$l_{j,B,r,s,t}^i : \langle A, p, r\rangle \rightarrow a \ \langle A_1, q, s\rangle \ \langle \check{B}, s, t\rangle \ \langle A_2, t, r\rangle \in P_1$$

where A is the topmost symbol of s_1^{i-1}. Alternatively, if s_2^{i-1} contains only variables then for all $X \in \Gamma \cup \{\epsilon\}$ and $r, s, t \in Q$include the production

$$l_{j,X,r,s,t}^i : \langle A, p, r\rangle \rightarrow a \ \langle A_1, q, s\rangle \ \langle \check{X}, s, t\rangle \ \langle A_2, t, r\rangle \in P_1$$

If the i level component is of the form

$$\left[s_1^{i-1} \mid x^i\right] \vdash \left[s_2^{i-1} \mid x^i\right]$$

then if the topmost symbol of s_2^{i-1} is a constant stack symbol $B \in \Gamma$ then for all $r, s, t \in Q$ include the production

$$l^i_{j,B,r,s,t} : \langle A, p, r\rangle \rightarrow a\langle B, \check{r}, q\rangle \in P_1$$

where A is the topmost symbol of s_1^{k-1}. Alternatively, if s_2^{k-1} contains only variables then for all $X \in \Gamma \cup \{\epsilon\}$ and $r, s, t \in Q$ include the production

$$l^k_{j,X,r,s,t} : \langle A, p, r\rangle \rightarrow a\langle X, \check{r}, q\rangle \in P_1$$

Finally the case where $i = 1$. The transitions has one of two forms.

$$\left[A \mid x^1\right] \vdash \left[A_1A_2 \mid x^1\right] \qquad \text{or} \qquad \left[A \mid x^1\right] \vdash \left[x^1\right]$$

In the first case, add the production

$$A \rightarrow L_j^2 A_1 A_2 \in P_i$$

In the second case, add the production

$$A \rightarrow L_j^2 \in P_i$$

In addition to the above productions, for each $1 < k \leq i$ we include productions for L_j^k giving $l^k_{j,A}$ for $k < i$ or $l^k_{j,A,p,q,r}$ for $k = i$ for all $A \in \Gamma \cup \{\epsilon\}$ and $p, q, r \in Q$. Rather than productions for the start symbol of each grammar, for the sake of simplicity let us assume that our grammars can have a set of start symbols rather than a single symbol. We let the entire nonterminal set of each grammar be in the start set.

At each stage in a computation of M^i, one of the rewrite rules, say the jth is used. Corresponding to the use of this rule, there will be i productions one in each of the grammars of C^i used. This is guaranteed by the use of labels one for each of the productions i components of the jth rewrite rule. Derivations of G_1 encode the topmost symbol on each of the $(i-1)$ order pushdowns. The derivation of G_2 that is controlling the derivation of G_1 corresponding to the kth of the $(i-1)$ level pushdowns will encode the topmost symbol of each of the $(i-2)$ level pushdowns on this pushdown, and so on, until eventually, G_i will encodes the entire remaining contents of the k_{i-1}th pushdown of the k_{i-2}th 2nd order pushdown of the ... of the k_1th $(i-1)$ order pushdown, for each $k_1, \ldots, k_{i-1}$. Each time M^i reads a symbol when using a rule j, the production for rule j in grammar G_1 will introduce that symbol into the derived string.

□

Chapter 4

Linear Context-Free Rewriting Systems

As a result of the observations made in Chapter 2, we outline how a class of Linear Context-Free Rewriting Systems (LCFRS's) may be defined and show how the string languages generated by these systems are semilinear and recognizable in polynomial time. Our goal is to capture the common properties shared by a number of grammar formalisms, without being unnecessarily restrictive. There are two ways in which we wish to constrain LCFRS's. Their derivation trees should be local sets, reflecting the fact that the derivation process should in a certain sense be context-free. Second, their composition operations should roughly speaking be linear and nonerasing. We begin this chapter by giving a more detailed description of LCFRS's. We then go on to show that they always generate semilinear languages, and that their recognition problem is in P. In Section 4.5 we show that the class of string languages generated by MCTAG's equals the class of languages generated by all LCFRS's. Sections 4.1, def-lcfr-b, semi-lcfr, and lcfr-in-P were presented in [86].

4.1 Generalized Context-Free Grammars

We define Generalized Context-Free Grammars (GCFG's), first discussed, though with a somewhat different definition, in [62].

Definition 4.1.1 A GCFG G is written as $G = (V, S, F, P)$, where

V is a finite set of variables

S is a distinguished member of V

F is a finite set of function symbols

P is a finite set of productions of the form

$$A \rightarrow f(A_1, \ldots, A_n)$$

where $n \geq 0$, $f \in F$, and $A, A_1, \ldots, A_n \in V$.

The set of terms, $T(G)$ derived from a GCFG, G is the set of all t such that $S \overset{*}{\underset{G}{\Longrightarrow}} t$ where the *derives* relation is defined as follows.

- $A \underset{G}{\Longrightarrow} f()$ if $A \rightarrow f()$ is a production.
- $A \overset{*}{\underset{G}{\Longrightarrow}} f(t_1, \ldots, t_n)$ if $A \rightarrow f(A_1, \ldots, A_n)$ is a production, and $A_i \overset{*}{\underset{G}{\Longrightarrow}} t_i$ for $1 \leq i \leq n$.

Notice that in GCFG's, rewriting choices during the derivation are independent of context. As we have seen in the previous discussion of Chapter 2, a GCFG will generate a set of trees that can be interpreted as derivation tree in various grammar formalisms. We wish to define a class of formalism for which this is true, called **Linear Context-Free Rewriting Systems.** By giving an interpretation for each of the functions in F, each term (tree) in $T(G)$ can be seen as encoding the derivation of some derived structure. For example, F may be functions over DAG's, in which case, each term will encode the composition of some derived DAG.

We will write the interpretation of some term $t \in T(G)$ in some formalism $\mathcal{F}$, as $[\![t]\!]_{\mathcal{F}}$. In order to illustrate some of the range of possibilities, we now consider several example of LCFRS's, giving one possible representations of their derivations.

Context-Free Grammars CFG's are string manipulation systems, thus the members of F will range over strings on some alphabet. For each zero arity function f, we let $[\![f()]\!]_{CFG} = w$ for some constant string of symbols w. For each function of n arguments, where $n \geq 1$, we let

$$[\![f(t_1, \ldots, t_n)]\!]_{CFG} = [\![t_1]\!]_{CFG} \circ \ldots \circ [\![t_n]\!]_{CFG}$$

i.e., the concatenation of the interpretation of its arguments.

Head Grammars HG's are a system that manipulates pairs of strings. For each zero arity function f, we let $[\![f()]\!]_{HG} = \langle w_1, w_2 \rangle$ for some constant strings w_1 and w_2. For each function of n arguments, where $n \geq 1$, we let

$$[\![f(t_1, \ldots, t_n)]\!]_{HG} = f_{HG}([\![t_1]\!]_{HG}, \ldots, [\![t_n]\!]_{HG})$$

where f_{HG} is C_i for some $1 \leq i \leq n$, or W in which case $n = 2$.

Tree Adjoining Grammars TAG's are tree manipulation systems, thus the members of F will range over labelled trees some of which have distinguished foot nodes. For each zero arity function f, we let $[\![f()]\!]_{TAG} = \gamma$, where γ is a complete elementary tree (i.e., has no OA nodes). For each function of n arguments, where $n \geq 1$, we let

$$\begin{aligned} & [\![f(t_1, \ldots, t_n)]\!]_{TAG} \\ = \ & \gamma[a_1, [\![t_1]\!]_{TAG}] \ldots [a_n, [\![t_n]\!]_{TAG}] \end{aligned}$$

γ is a tree in which at addresses $a_1, \ldots, a_n$ trees $[t_1]_{TAG}, \ldots, [t_n]_{TAG}$ are adjoined to produce $[f(t_1, \ldots, t_n)]_{TAG}$.

Multicomponent Tree Adjoining Grammars MCTAG's are tree manipulation systems, and we define the composition operations on finite sequences of trees, rather than tree sets. For each zero arity function f, we let $[f()]_{MCTAG} = \langle \gamma_1, \ldots, \gamma_k \rangle$ where each γ is a complete elementary tree. For each function of n arguments, where $n \geq 1$, we let

$$\begin{aligned} [f(t_1, \ldots, t_n)]_{MCTAG} = \\ \langle \gamma_1, \ldots, \gamma_k \rangle \quad & [(i_{1,1}, a_{1,1}), \ldots, (i_{1,m_1}, a_{1,m_1}), [t_1]_{MCTAG}] \cdots \\ & [(i_{n,1}, a_{n,1}), \ldots, (i_{n,m_1}, a_{n,m_n}), [t_n]_{MCTAG}] \end{aligned}$$

This indicates that the tree sequence $[f(t_1, \ldots, t_n)]_{MCTAG}$ is derived from the sequence $\langle \gamma_1, \ldots, \gamma_k \rangle$ such that the pth tree in the sequence $[t_q]_{MCTAG}$ is adjoined into $\gamma_{i_{q,p}}$ at the node with address $a_{q,p}$.

4.2 Composition Operations

In each of the LCFRS's that we have given, the functions (combining strings, trees, or sequences of strings and trees) share certain constrained properties. Although it would be a desirable accomplishment, we will not attempt to completely characterize the entire class of such functions that will be permitted. This would be difficult since we are considering formalisms with arbitrary structures. Instead, we will give two restrictions on the functions. We would like these restrictions to ensure that the functions do not "copy", "erase", or "restructure" unbounded components of their arguments. The result of composing any two structures should be a structure whose "size" is the sum of its constituents plus some constant.

Every intermediate structure that a grammar derives contributes some terminals to the string that is yielded by the structure that is finally derived. However, the symbols in the yield of an intermediate structure do not necessarily form a continuous substring of the final string. In general, though, we can write the yield of an intermediate structure as a finite sequence of substrings of the final string. If $[\![t]\!]_{\mathcal{F}}$ is some intermediate structure produced by some formalisms $\mathcal{F}$ (i.e., t is a subterm of some $t' \in T(G)$) then let there be a function $\phi_{\mathcal{F}}$ giving the yield of structures, such that $\phi_{\mathcal{F}}([\![t]\!]_{\mathcal{F}}) = \langle w_1, \ldots, w_k \rangle$, where $\langle w_1, \ldots, w_k \rangle$ is a sequence of terminal strings.

> **Restriction 1:** Given a LCFRS, $\mathcal{F}$, for each $f \in F$ there will be a unique yield function (which we denote $\bar{f}_{\mathcal{F}}$) such that if
>
> $$\phi_{\mathcal{F}}([\![t_1]\!]_{\mathcal{F}}) = \langle w_{1,1}, \ldots, w_{1,k_1} \rangle,$$
> $$\ldots, \quad \phi_{\mathcal{F}}([\![t_n]\!]_{\mathcal{F}}) = \langle w_{n,1}, \ldots, w_{n,k_n} \rangle$$
>
> and
>
> $$\phi_{\mathcal{F}}([\![f(t_1, \ldots, t_n)]\!]_{\mathcal{F}}) = \langle w_1, \ldots, w_k \rangle$$
>
> then
>
> $$\bar{f}_{\mathcal{F}}(\langle w_{1,1}, \ldots, w_{1,k_1} \rangle, \ldots, \langle w_{n,1}, \ldots, w_{n,k_n} \rangle) = \langle w_1, \ldots, w_k \rangle$$
>
> Thus, the sequence of substrings that is the yield of an intermediate structure is independent of how that structure is used later in the derivation.

As we have stated, the composition operations should be "size" preserving. Thus, with respect to the yield of the structures being manipulated, we should expect the composition operations to do no more than reorder their arguments and insert a bounded number of additional terminals.

Restriction 2: It will be possible to define each $\bar{f}_{\mathcal{F}}$ with an equation of the following form.

$$\bar{f}_{\mathcal{F}}(\langle x_{1,1},\ldots,x_{1,m_1}\rangle,\ldots,\langle x_{n,1},\ldots,x_{n,m_n}\rangle) = \langle t_1,\ldots,t_m\rangle$$

where $n \geq 0$, each t_i is a string of variables (x's), and some finite number terminal symbols. The equations are **regular** (all the variables appearing on one side appear on the other) and **left** and **right linear** (the variables appear only once on the left and right).

In order to study the string languages generated by members of the class of LCFRS's, we can view their grammars as follows. Let $\mathcal{F}$ be some LCFRS, a grammar of $\mathcal{F}$ can be expressed as (G, E) where $G = (V, S, F, P)$ is a GCFG, and E a finite set of equations, E, defining each function $\bar{f}_{\mathcal{F}}$ where $f \in F$ as given in Restriction 2.

The set of strings $L(G, E)$ defined by such a grammar is given as follows.

$$L(G,E) = \{\, w_1 \ldots w_m \mid S \underset{G,E}{\overset{*}{\Longrightarrow}} \langle w_1,\ldots,w_m\rangle \,\}$$

where the *derives* relation is defined as follows. $A \underset{G,E}{\overset{*}{\Longrightarrow}} \langle w_1,\ldots,w_m\rangle$ if one of the following holds.

- $A \to f() \in P$ and $\bar{f}_{\mathcal{F}}() = \langle w_1,\ldots,w_m\rangle \in E$.
- $A \to f(A_1,\ldots,A_n) \in P$,

$$A_1 \underset{G,E}{\overset{*}{\Longrightarrow}} \langle w_{1,1},\ldots,w_{1,m_1}\rangle,\ldots,A_n \underset{G,E}{\overset{*}{\Longrightarrow}} \langle w_{n,1},\ldots,w_{n,m_n}\rangle$$

$$\bar{f}_{\mathcal{F}}(\langle x_{1,1},\ldots,x_{1,m_1}\rangle,\ldots,\langle x_{n,1},\ldots,x_{n,m_n}\rangle) = \langle t_1,\ldots,t_m\rangle \in E$$

and w_i for $1 \leq i \leq m$ results from substituting all occurrences of $x_{p,q}$ in t_i by $w_{p,q}$, for all $1 \leq p \leq n$ and $1 \leq q \leq m_p$.

Notice that we have not allowed any erasing, not even to a bounded extent. This may turn out to be a restriction that we wish to overcome. However, at this point, all of the systems that we have considered do not allow any erasing.

4.3 Semilinearity of LCFRL's

Theorem 4.3.1 If L is a language generated by a grammar of some formalism that is a LCFRS, then L is a semilinear language.

Semilinearity and the closely related constant growth property (a consequence of semilinearity) have been discussed in the context of grammars for natural languages by Joshi [36] and Berwick and Weinberg [11]. Roughly speaking, a language, L, has the property of semilinearity if the number of occurrences of each symbol in any string is a linear combination of the occurrences of these symbols in some fixed finite set of strings. Thus, the length of any string in L is a linear combination of the length of strings in some fixed finite subset of L, and thus L is said to have the constant growth property. The definition is given in Chapter 1. Although this property is not structural, it depends on the structural property that sentences can be built from a finite set of clauses of bounded structure as noted by Joshi [36].

The property of semilinearity is concerned only with the occurrence of symbols in strings and not their order. Thus, any language that is letter equivalent to a semilinear language is also semilinear. Two strings are letter equivalent if they contain equal number of occurrences of each terminal symbol, and two languages are letter equivalent if every string in one language is letter equivalent to a string in the other language and vice-versa. Since every CFL is known to be semilinear [60], in order to show semilinearity of some language, we need only show the existence of a letter equivalent CFL.

Given some fixed alphabet $\Sigma = \{a_1, \ldots, a_n\}$, let $\psi(w)$ (the Parikh mapping of w) be defined such that $\psi(w) = (i_1, \ldots, i_n)$ where each i_j, $1 \leq j \leq n$ is an integer equal to the number of occurrences of the symbol a_{i_j} in the string w. The Parikh mapping $\psi(L)$, of a language L is the set of all vectors $\psi(w)$ such that $w \in L$. For each language L generated by a grammar of a LCFRS, we will show that there is a CFL, L', such that $\psi(L) = \psi(L')$.

Let $\mathcal{F}$ be some LCFRS, consider a grammar of $\mathcal{F}$ where GCFG is $G = (V, S, F, P)$, and E contains the definition of each function $\bar{f}_{\mathcal{F}}$ such that $f \in F$. We now define a CFG, $G' = (V, V_T, S, P')$ such that $\psi(L(G, E)) = \psi(G')$.

- If $A \rightarrow f() \in P$ and $\bar{f}_{\mathcal{F}}() = \langle w_1, \ldots, w_m \rangle \in E$ then

$$A \rightarrow w_1 \ldots w_m \in P'$$

- If $A \rightarrow f(A_1, \ldots, A_n) \in P$ and

$$\begin{aligned} &\bar{f}_{\mathcal{F}}(\langle x_{1,1}, \ldots, x_{1,m_1} \rangle, \\ \ldots, \quad &\langle x_{n,1}, \ldots, x_{n,m_n} \rangle) = \langle t_1, \ldots, t_m \rangle \in E \end{aligned}$$

  then

$$A \rightarrow A_1 \ldots A_n w \in P'$$

  where w is the concatenation of all occurrences of the terminal symbols appearing in some t_i, for $1 \leq i \leq m$, i.e., the bounded number of new symbols introduced by $\bar{f}_{\mathcal{F}}$.

It can be shown by induction on the length of derivations that

$$A \overset{*}{\underset{G,E}{\Longrightarrow}} \langle w_1, \ldots, w_m \rangle$$

where $\psi(w_1 \ldots w_m) = v$ if and only if $A \overset{*}{\underset{G'}{\Longrightarrow}} w$ where $\psi(w) = v$.

4.4 LCFRL's in P

Theorem 4.4.1 If L is a language generated by a grammar of some formalism that is a LCFRS, then L can be recognized in polynomial time on a turing machine.

We now turn our attention to the recognition of string languages generated by these formalisms (LCFRL's). This can be seen as a special case of a result in [67]. Although embedding this version of LCFRS's in a framework developed by Rounds in [67] is straightforward, our motivation was to capture properties shared by a family of grammatical systems and generalize them defining a class of related formalisms.

4.4.1 Alternating Turing Machines

We use Alternating Turing Machines [13] to show that polynomial time recognition is possible for the languages discussed in Section 4.4.2. An ATM has two types of states, existential and universal. In an existential state an ATM behaves like a nondeterministic TM, accepting if one of the applicable moves leads to acceptance; in an universal state the ATM accepts if all the applicable moves lead to acceptance. An ATM may be thought of as spawning separate processes for each applicable move. A k-tape ATM, M, has a read-only input tape and k read-write work tapes. A *step* of an ATM consists of reading a symbol from each tape and optionally moving each head to the left or right one tape cell. A **configuration** of M consists of a state of the finite control, the nonblank contents of the input tape and k work tapes, and the position of each head. The **space** of a configuration is the sum of the lengths of the nonblank tape contents of the k work tapes. M works in space $S(n)$ if for every string that M accepts no configuration exceeds space $S(n)$. It has been shown in [13] that if M works in space $\log n$ then there is a deterministic TM which accepts the same language in polynomial time. In the next section, we show how an ATM can accept the strings generated by a grammar in a LCFRS formalism in logspace, and hence show that each family can be recognized in polynomial time.

4.4.2 Recognition of LCFRS's

Given a grammar (G, E), where $G = (V, S, F, P)$, belonging to some LCFRS, $\mathcal{F}$, we describe the operations of an ATM, M, recognizing $L(G, E)$. M performs a top-down recognition of an input $a_1 \ldots a_n$ in logspace.

A substring of the input string is encoded by a pair of integers marking the two end positions of the substring. These positions range from 0 to n where n is the length of the input string, and can therefore be encoded in $O(\log n)$ space. We encode a m-tuple $\langle w_1, \ldots, w_m \rangle$ of substrings using $2m$ tapes. Let k be the maximum number of components of any tuple on the right-hand-side of an equation in E. At any stage in the derivation, we may have to store $3k$ pairs of integers requiring $6k$ tapes. In addition to the tapes for holding these indices, M requires working tapes for indices recording the position of terminals appearing in the equations in E, and requires one additional tape for temporary storage of indices.

Without loss of generality, we will assume that every function $\bar{f}_{\mathcal{F}}$ is defined so that it is either a constant function or a binary function whose arguments and result are k tuples. The tuples can be "filled" with empty strings, if necessary.

We will describe how M will recognize an input string $a_1 \ldots a_n$. Starting in an existential state, the machine will nondeterministically choose any one of the ways of spanning the input string with k substrings, storing these on the first $2k$ tapes, and moving into an existential state q_S.

At some arbitrary point in the derivation, M will be in a configuration intended to determine whether the k substrings $w_1, \ldots, w_k$ can be derived from some symbol A at a specific k positions in the input. M will be in an existential state q_A, with integers i_1 and i_2 representing w_i in the $(2i-1)^{th}$ and $2i^{th}$ work tape, for $1 \leq i \leq k$.

For each rule $A \rightarrow f(B, C) \in P$ and equation

$$\bar{f}_{\mathcal{F}}(\langle x_1, \ldots, x_k \rangle, \langle y_1, \ldots, y_k \rangle) = \langle t_1, \ldots, t_k \rangle \in E$$

M must break $w_1, \ldots, w_k$ into the unbounded substrings $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$ and a finite number of terminals in specified positions. M spawns as many processes as there are ways of breaking up $t_1, \ldots, t_k$ and rules with A on their left-hand-side. Each spawned process must check if $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$ can be derived from B and C, respectively, and that each of the terminals occurring in some t_i appear in the correct position in the input. To do this, the x's and y's are stored in the next $4k$ tapes, and the positions of any terminals appearing in the equation are stored in the additional tapes. The identities of the nonterminals and terminals deriving the various substrings that are being stored on the working tapes can be encoded in the finite state control. The "calculation" of the new indices that result from the use of an equation will involve a series of moves. The pair of indices i_1 and i_2, encoding each t_i will result in a number of new pairs of indices that as a group "span" the complete gap from i_1 to i_2. Each of this group of new pairs of indices is placed on a pair of tapes according to the equation that has been used.

Once each of the t_i's has been considered, M goes to a universal state which has succeeding states checking that B derives $x_1, \ldots, x_k$, C derives $y_1, \ldots, y_k$, and each terminal a to appear in the specified position. One successor process will put M in the existential state q_B with the indices encoding $x_1, \ldots, x_k$ moved onto the first $2k$ tapes. Another succeeding state will put M in the existential state q_C with the indices encoding $y_1, \ldots, y_k$ moved onto the first $2k$ tapes.

For rules $A \rightarrow f() \in P$ such that $\bar{f}_{\mathcal{F}}() = \langle w_1, \ldots w_k, \rangle \in E$ where each w is a constant string, M must enter a universal state and check that each of the k constant substrings are in the appropriate place (as determined by the contents of the first $2k$ work tapes) on the input tape.

4.4.3 Proper containment in P

The class of languages generated by LCFRS's is a proper subset of P. There are languages in P that are not semilinear and can not be generated by any LCFRS. For example, the language $\{ a^{2^n} \mid n \geq 0 \}$ is not a semilinear language and can be recognized in polynomial time. A Turing Machine recognizing this language could read the input from left to right checking that each symbol is an a, and counting up how many symbols it has read, storing the length of the input in binary on a work tape. Once the right end marker is reached the Turing Machine must check that the number stored on the work tape is 10^n for some $n \geq 0$, i.e., that the length of the input is 2^n.

4.5 LCFRL's and MCTAL's

In this section we show that the class of languages generated by MCTAG's is weakly equivalent to the union of languages generated by LCFRS's (LCFRL's). We also show that a generalization of CFG's (MCCFG's) is also equivalent to MCTAG's.

We will not address the question of whether MCTAG's are strongly equivalent to every LCFRS. Normally strong equivalence is defined in terms of tree sets, however, this will not be possible in the case of LCFRS's which need not generate object level tree sets. It would be interesting to investigate whether there exist LCFRS's with object level tree sets that can not be produced by any MCTAG.

4.5.1 MCTAL's $\subseteq$ LCFRL's

We show that for each $k \geq 1$, the class of MCTAG's whose longest elementary tree sequence is no longer than k, form a LCFRS. This shows that MCTAL's are included in LCFRL's.

Consider some MCTAG whose longest tree sequence is k. We will show that this grammar can be expressed in such a way that the criterion for membership of LCFRS's are met. As we have show in Section 4.1, it follows directly from the definition of the formalism that we can represent MCTAG derivation by trees generated by a CFG. It is also clear that the calculation of the yield of any intermediate structure is independent of derivational context (Restrictions 1 of Section 4.2). It remains to be shown that the composition operations conform to the following restriction.

> For each composition operation f, $\bar{f}_{MCTAG}$ can be defined with an equation of the following form.
>
> $$\bar{f}_{MCTAG}(\langle x_{1,1}, \ldots, x_{1,m_1}\rangle, \ldots, \langle x_{n,1}, \ldots, x_{n,m_n}\rangle) = \langle t_1, \ldots, t_m\rangle$$
>
> where $n \geq 0$, each t_i is a string of variables (x's), and some finite number terminal symbols, the equations are regular and left and right linear (the variables appear only once on the left and right).

Each composition function f mentioned in the CFG encoding the MCTAG derivations, corresponds to one of the ways that n derived tree sequences can be multi-adjoined into an elementary tree sequence. For an arbitrary such f will describe the equation for $\bar{f}_{MCTAG}$, and show that it conforms to the above restriction.

The yield of a derived auxiliary tree sequence $\langle \gamma_1, \ldots, \gamma_m \rangle$ will be the following $2m$ tuple.

$$\phi_{MCTAG}(\langle \gamma_1, \ldots, \gamma_m \rangle) = \langle u_1, u_1', \ldots, u_m, u_m' \rangle$$

where u_i and u_i' are the yields to the left and right or the foot node of γ_i, respectively. In the case of a derived initial tree, γ, $\phi_{MCTAG}(\gamma) = u$ where u is the frontier of γ.

We consider an arbitrary multi-component adjunction. Our only assumption will be that we are adjoining into an *auxiliary* tree sequence, it should be clear how this case can be modified to deal with initial trees. Let f correspond to adjoining members of the n derived auxiliary tree sequences

$$\langle \gamma_{1,1}, \ldots, \gamma_{1,m_1} \rangle, \ldots, \langle \gamma_{n,1}, \ldots, \gamma_{n,m_n} \rangle$$

into the elementary tree sequence $\langle \gamma_1, \ldots, \gamma_m \rangle$.

By assumption, we know that $m, m_1, \ldots, m_n \leq k$. Our goal is to describe each t_i, and t_i' in the following equation.

$$\begin{aligned}&\bar{f}_{MCTAG}(\left\langle x_{1,1}, x_{1,1}', \ldots, x_{1,m_1}, x_{1,m_1}' \right\rangle, \ldots, \left\langle x_{n,1}, x_{n,1}', \ldots, x_{n,m_n}, x_{n,m_n} \right\rangle)\\ &= \langle t_1, t_1', \ldots, t_m, t_m' \rangle\end{aligned}$$

In this equation, t_i and t_i' should give us the frontier to the left and right of the foot node of γ_i after this multi-component adjunction has taken place. $x_{i,j}$ and $x_{i,j}'$ are variables that will match the left and right frontier of the jth tree in the ith tree sequence, i.e., $\gamma_{i,j}$.

It is clear that this equation will be regular and linear. Regularity (the same variables appear on each side) follows from the restriction in MCTAG's that every tree in a tree sequence is involved in the adjunction. Linearity (variables only appear once on each side) follows from the restriction that each tree in a tree sequence is only adjoined into a single place.

We now describe the terms t_i, and t'_i giving the frontier of γ_i. We do this by defining two functions f_l and f_r giving the terms corresponding to the yield of subtrees of tree in the sequence $\langle \gamma_1, \ldots, \gamma_m \rangle$ such that $f_l(\gamma_i) = t_i$ and $f_r(\gamma_i) = t'_i$. These functions are defined by induction on the height of γ_i. Suppose that γ_i is a single node.

- If it is the foot node and none of the auxiliary tree are adjoined at this node then $f_l(\gamma) = f_r(\gamma) = \epsilon$.

- Suppose that $\gamma_{p,q}$ was adjoined at this node, in this case $f_l(\gamma_i) = x_{p,q}$ and $f_r\gamma_i = x'_{p,q}$.

- If this node is labeled by a (a terminal or the empty string) then let $f_l(\gamma)_i = a$ and $f_r(\gamma) = \epsilon$.

Consider that γ_i has height h, and that the root of γ_i has r subtrees $\gamma_{i_1}, \ldots, \gamma_{i_r}$, and the root node is in γ_{i_p}.

- If none the auxiliary trees are adjoined at the root of γ_i then let

$$f_l(\gamma_i) = f_l(\gamma_{i_1}) f_r(\gamma_{i_1}) \circ \ldots \circ f_l(\gamma_{i_p})$$

  and

$$f_r(\gamma_i) = f_r(\gamma_{i_p}) \circ \ldots \circ f_l(\gamma_{i_r}) f_r(\gamma_{i_r})$$

  i.e., concatenate the yield of each subtree.

- If $\gamma_{p,q}$ was adjoined at the root of γ_i, then let

  $$f_l(\gamma_i) = x_{p,q} f_l(\gamma_{i_1}) f_r(\gamma_{i_1}) \circ \ldots \circ f_l(\gamma_{i_p})$$

  and

  $$f_r(\gamma_i) = f_r(\gamma_{i_p}) \circ \ldots \circ f_l(\gamma_{i_r}) f_r(\gamma_{i_r}) x'_{p,q}$$

  The variable $x_{p,q}$ (or $x'_{p,q}$) corresponds to the yield to the left (or right) of the root node of $\gamma_{p,q}$.

4.5.2 LCFRL's $\subseteq$ MCTAL's

Consider a LCFRS $\mathcal{F}$ that involves manipulation of k-tuples with a set of operations F producing languages over the terminal alphabet Σ. Without loss of generality we will assume that every operations in F is over exactly k-tuples. Given a grammar $G = (V, S, P, F)$ of this formalism $\mathcal{F}$, we describe a MCTAG, G' such that $L(G) = L(G')$.

- Suppose that P contains

  $$A \rightarrow f()$$

  where $\bar{f}_{\mathcal{F}}() = \langle w_1, \ldots, w_k \rangle$.

  Corresponding to this production the MCTAG, G' will contain an auxiliary tree sequence containing k auxiliary trees, as follows.

  For each $1 \leq i \leq k$, if $w_i = a_{i,1} \ldots a_{i,m_i}$ then the ith tree in the sequence will be the following tree. It is rooted by an NA node labeled $\langle A, i \rangle$ with $|w_i| + 1$ children. The first $|w_i|$ of the children are labeled by the terminals $a_{i,1}, \ldots, a_{i,m_i}$, and the remaining child is the foot node and is a NA node labeled by the nonterminal $\langle A, i \rangle$.

be rewritten by one of the production for that symbol. It is not possible to ensure that a certain restricted combination of productions are used. Furthermore, there is no distinction between terminals and nonterminals, and *every* sentential form is in the language generated. As a result, 0L-systems are unable to generate some extremely simple languages such as $\{ a, aa \}$.

While 0L-systems resemble MCCFG's because they involve simultaneous rewriting, there is another system called matrix grammars [1] that shares a different property with MCCFG's. Matrix grammars contain *sequences* of productions that must be used as a group. The difference between MCCFG's and matrix grammars is that the sequences of productions in matrix grammars are not used simultaneously, but one immediately after another, and in a particular order.

4.6 Summary

In Chapter 2 we studied the structural descriptions (tree sets) that can be assigned by various grammatical systems, and classified these formalisms on the basis of two features: path complexity; and path independence. We contrasted formalisms such as CFG's, HG's, TAG's and MCTAG's, with formalisms such as IG's and unificational systems such as LFG's and FUG's. In order to observe the similarity between these constrained systems, it is crucial to abstract away from the details of the structures and operations used by the system. The similarities become apparent when they are studied at the level of *derivation structures*: derivation tree sets of CFG's, HG's, TAG's, and MCTAG's are all local sets.

In this chapter, we outlined the definition of a family of constrained grammatical formalisms, called Linear Context-Free Rewriting Systems. This family represents an attempt to generalize the properties shared by CFG's, HG's, TAG's, and MCTAG's. Like HG's, TAG's, and MCTAG's, members of LCFRS can manipulate structures more complex than terminal strings and use composition operations that are more complex that concatenation. We place certain restrictions on the composition operations of LCFRS's, restrictions that are shared by the composition operations of the constrained grammatical systems that we have considered. The operations must be linear and nonerasing, i.e., they can not duplicate or erase structure from their arguments. Notice that even though IG's and LFG's involve CFG-like productions, they are (linguistically) fundamentally different from CFG's because the composition operations need not be linear. By sharing stacks (in IG's) or by using nonlinear equations over f-structures (in FUG's and LFG's), structures with unbounded dependencies between paths can be generated.

Having defined LCFRS's, we established the semilinearity (and hence constant growth property) of the languages generated. In considering the recognition of these languages, we were forced to be more specific regarding the relationship between the structures derived by these formalisms and the substrings they span. We insisted that each structure dominates a bounded number of (not necessarily adjacent) substrings. The composition operations are mapped onto operations that use concatenation to define the substrings spanned by the resulting structures. We showed that any system defined in this way can be recognized in polynomial time. It was previously known that CFG's, HG's, and TAG's can be recognized in polynomial time since polynomial time algorithms exist for each of these formalisms. A corollary of the result of Section 4.4.2 is that polynomial time recognition of MCTAG's is possible. In Section 4.5 we showed that the class of languages generated by MCTAG's is equal to the class of languages generated by LCFRS's.

- Suppose that P contains

$$A \rightarrow f(A_1, \ldots, A_n)$$

where

$$\bar{f}_{\mathcal{F}}(\langle x_{1,1}, \ldots, x_{1,k}\rangle, \ldots, \langle x_{n,1}, \ldots, x_{n,k}\rangle) = \langle t_1, \ldots, t_k\rangle$$

Corresponding to this production the MCTAG, G' will contain an auxiliary tree sequence containing k auxiliary trees.

For each $1 \leq i \leq k$, suppose that $t_i = a_1 x_1 a_2 \ldots a_r x_r a_{r+1}$ where each $a_j \in \Sigma \cup \{\epsilon\}$ for $1 \leq j \leq r+1$, and each

$$x_j \in \{x_{1,1}, \ldots, x_{1,m_1}, \ldots, x_{n,1}, \ldots, x_{n,m_n}\}$$

for $1 \leq j \leq r$. The ith tree in the sequence has its root be an NA node labeled by $\langle A, i\rangle$. The root will have $2r+2$ children.

The $2r+2$th child is an NA node that is the foot node (labeled by $\langle A, i\rangle$).

For $1 \leq j \leq r+1$ the $2j-1$th child is labeled by a_j.

For $1 \leq j \leq r$ the $2j$th child will be labeled by the nonterminal $\langle A_p, q\rangle$ if $x_j = x_{p,q}$, and each of these nodes has an OA constraint and a single child labeled by ϵ.

It is straightforward to show by induction on the length of the derivation in G that when ever a nonterminal A derives a k-tuple $\langle w_1, \ldots, w_k\rangle$, there will be a derived auxiliary tree set containing k auxiliary trees with root labeled by $\langle A, 1\rangle, \ldots, \langle A, k\rangle$ and frontiers $w_1\langle A, 1\rangle, \ldots, w_k\langle A, k\rangle$, respectively. There is one initial tree with root labeled S having k children labeled by $\langle S, 1\rangle, \ldots, \langle S, k\rangle$. Each of these k nodes has an OA constraint, and a single child labeled by ϵ.

4.5.3 Multi-Component CFG's

It is worth noting that in the construction given here the adjunction operation is in effect merely simulating tree substitution. Since adjunction only occurs at nodes which have a single child labeled by ϵ, we could instead have omitted the node labeled ϵ and omitted the foot nodes of each auxiliary tree, and used substitution rather than adjunction. Each tree would then have only two levels of nodes and NA constraints at the root node. Each tree would therefore be equivalent to a context-free production, and a tree sequence equivalent to a sequence of productions. We could then have performed multicomponent substitution at each of the resulting frontier node labeled by a nonterminal. Therefore, MCTAG's and Multicomponent CFG's (a system in which a sequence of CFG productions must be used concurrently) are weakly equivalent[1]. The derivation of a MCCFG should be performed "inside-out". It would begin with sequences containing productions all of whose right-hand sides contained only terminals. At each subsequence stage of the derivation all of the productions in several derived sequences of productions are simultaneously used to rewrite all of the nonterminals on the right-hand sides of the productions in another sequence to produce a new derived sequences. The derivation could end when a derived sequence of length 1 was produced containing a production whose left-hand side was the start symbol.

Such a system is somewhat similar to Lindenmayer systems (L-systems), particularly 0L-systems [53]. As with the system we have described 0L-systems involve simultaneous rewriting with context-free productions. However, at each stage in derivations of 0L-systems, every occurrence of a symbol in a sentential form must

[1]The extra power of the adjunction operation would be reflected in the fact that MCTAG's would in general require smaller tree sequences than MCCFG's to generate the same string languages.

It appears that HG's (and TAG's) can not generate all of the languages that can be generated by arbitrary LCFRS's manipulating 2-tuples. Consider the following binary operation over tuples.

$$F(\langle u_1, u_2\rangle, \langle v_1, v_2\rangle) = \langle u_1v_1, u_2v_2\rangle$$

Consider an extension to HG's in which this operation can also be used together with the usual concatenation and wrapping operations. Using this extended set of operations we can generate the language $L = \{\, a_1^n b_1^n a_2^m b_2^m c_1^n d_1^n c_2^m d_2^m \mid n, m \geq 0 \,\}$ with the grammar below.

$$\begin{array}{ll} S \rightarrow F(S_1, S_2) & \\ S_1 \rightarrow W(\langle a_1, d_1\rangle, T_1) & S_2 \rightarrow W(\langle a_2, d_2\rangle, T_2) \\ S_1 \rightarrow C_1(\langle \epsilon, \epsilon\rangle) & S_2 \rightarrow C_1(\langle \epsilon, \epsilon\rangle) \\ T_1 \rightarrow W(S_1, \langle b_1, c_1\rangle) & T_2 \rightarrow W(S_2, \langle b_2, c_2\rangle) \end{array}$$

Unfortunately, the pumping lemma for TAL's given in [82] (Theorem 5.5.1) is too weak to show that this language is not a TAL. However, we believe that this language can not be generated by TAG's or HG's, and that a sufficiently strong pumping lemma (perhaps one analogous to that of Ogden [56]) it would be possible to show this.

Chapter 5

Combinatory Categorial Grammars

In this section, we examine Combinatory Categorial Grammars (CCG's), an extension of Classical Categorial Grammars [5] developed by Steedman and his collaborators [2, 78, 75, 76, 77]. Classical Categorial Grammars are known to be weakly equivalent to CFG's [7], and the main result here is that under a certain definition, CCG's are weakly equivalent to TAG's, HG's, and LIG's. We prove this by showing in Section 5.2 that Combinatory Categorial Languages (CCL's) are included in Linear Indexed Languages (LIL's), and that Tree Adjoining Languages (TAL's) are included in CCL's. We also presented this result in [88].

On the basis of their weak equivalence with TAG's, and HG's, it appears that CCG's should be classified as a mildly context-sensitive grammar formalism. In Section 4 we consider whether CCG's should be included in the class of LCFRS's. The derivation tree sets traditionally associated with CCG's have Context-free path sets, and are similar to those of LIG's, and therefore differ from those of LCFRS's.

This does not, however, rule out the possibility that there may be alternative ways of representing the derivation of CCG's that will allow for their classification as LCFRS's.

Extensions to CCG's have been considered that enable them to compare two unbounded structures (for example, in [78]). It has been argued that this may be needed in the analysis of certain coordination phenomena in Dutch. In Section 5.3 we discuss how these additional features increase the power of the formalism. In so doing, we also give an example demonstrating that the Parenthesis-free Categorial Grammar formalism [23, 22] is more powerful than CCG's as defined here. Extensions to TAG's (Multicomponent TAG) have been considered for similar reasons. However, we will not investigate the relationship between the extension of CCG's and Multicomponent TAG.

It is known [82] that the complexity of TAL recognition is $O(n^6)$. Thus, a corollary of our result is that this is also a property of CCL's. Although there has been previous work [90, 59] on the parsing of CCG's, they have not suggested a specific upper bound on recognition.

5.1 Definition of CCG's

Combinatory Categorial Grammar (CCG), as defined here, is the most recent version of a system that has evolved in a number of papers [2, 78, 75, 76, 77]. In this section we first define CCG's, and then show that the class of string languages generated by CCG's is equal to the languages generated by TAG's (and LIG's).

Definition 5.1.1 A CCG, G, is denoted by (V_T, V_N, S, f, R) where

V_T is a finite set of terminals (lexical items),

V_N is a finite set of nonterminals (atomic categories),

V_N and V_T are disjoint sets

S is a distinguished member of V_N,

f is a function that maps elements of $V_T \cup \{\epsilon\}$ to finite subsets of $C(V_N)$, the set of categories[1], where

> $V_N \subseteq C(V_N)$ and if $c_1, c_2 \in C(V_N)$ then $(c_1/c_2) \in C(V_N)$ and $(c_1\backslash c_2) \in C(V_N)$.

R is a finite set of combinatory rules.

There are four types of combinatory rules, which involve object level variables $x, y, z_1, \ldots$ over $C(V_N)$, and each $|_i$ in the rules belongs to the set $\{ \backslash, / \}$.

1. forward application:

$$(x/y) \quad y \rightarrow x$$

2. backward application:

$$y \quad (x\backslash y) \rightarrow x$$

3. generalized forward composition for some $n \geq 1$:

$$(x/y) \quad (\ldots(y|_1 z_1)|_2 \ldots |_n z_n) \rightarrow (\ldots(x|_1 z_1)|_2 \ldots |_n z_n)$$

4. generalized backward composition for some $n \geq 1$:

$$(\ldots(y|_1 z_1)|_2 \ldots |_n z_n) \quad (x\backslash y) \rightarrow (\ldots(x|_1 z_1)|_2 \ldots |_n z_n)$$

Restrictions can be associated with the use of the combinatory rules in R. These restrictions take the form of constraints on the instantiations of variables in the rules. These can be constrained in two ways.

[1]Note that f can assign categories to the empty string, ϵ, though, to our knowledge, this feature has not been employed in the linguistic applications of CCG.

1. The initial nonterminal of the category to which x is instantiated can be restricted.

2. The entire category to which y is instantiated can be restricted.

Derivations in a CCG involve the use of the combinatory rules in R. Let the *derives* relation be defined as follows.

$$\alpha c \beta \underset{G}{\Longrightarrow} \alpha c_1 c_2 \beta$$

if R contains a combinatory rule that has $c_1 c_2 \rightarrow c$ as an instance, and α and β are (possibly empty) strings of categories. The string languages, $L(G)$, generated by a CCG, G, is defined as follows.

$$\{a_1 \ldots a_n \mid S \overset{*}{\underset{G}{\Longrightarrow}} c_1 \ldots c_n, c_i \in f(a_i), a_i \in V_T \cup \{\epsilon\}, 1 \leq i \leq n\}$$

Although there is no type-raising rule in the above formulation, its effect can be achieved to a limited extent since f can assign type-raised categories to lexical items. This is the scheme employed in Steedman's recent work.

5.2 Weak Generative Capacity

In this section we show that CCG's are weakly equivalent to TAG's, HG's, and LIG's. We do this by showing the inclusion of CCL's in LIL's, and the inclusion of TAL's in CCL's. It is know that TAG and LIG are equivalent [82], and that TAG and HG are equivalent [87, 89]. Thus, the two inclusions shown here imply the weak equivalence of all four systems.

5.2.1 CCL's $\subseteq$ LIL's

We describe how to construct a LIG, G', from an arbitrary CCG, G such that G and G' are equivalent. Let us assume that by default the slashes associate to the left. Thus, categories are written without parentheses, unless they are needed to override the left associativity of the slashes.

> A category c is **minimally parenthesized** if and only if one of the following holds.
>
> $c = A$ for $A \in V_N$
>
> $c = (A|_1 c_1|_2 \dots |_n c_n)$, for $n \geq 1$, where $A \in V_N$ and each c_i is minimally parenthesized.

It will be useful to be able to refer to the **components** of a category, c. We first define the immediate components of c.

> when $c = A$ the immediate component is A,
>
> when $c = (A|_1 c_1|_2 \dots |_n c_n)$ the immediate components are $A, c_1, \dots, c_n$.

The components of a category c are its immediate components, as well as the components of its immediate components. The immediate components are the categories arguments. Thus, $c = (A|_1 c_1|_2 \dots |_n c_n)$ is a category that takes has n arguments of category $c_1, \dots, c_n$ to give the **target** category A.

Although in CCG's there is no bound on the number of categories that are derivable during a derivation (categories resulting from the use of a combinatory rule), there is a bound on the cardinality of the set of all *components* that derivable categories may have. This would no longer hold if unrestricted type-raising were allowed during a derivation.

Let the set $D_C(G)$ be defined as follows.

$c \in D_C(G)$ if c is a component of c' where $c' \in f(a)$ for some $a \in V_T \cup \{\epsilon\}$.

Clearly for any CCG, G, $D_C(G)$ is a finite set. $D_C(G)$ contains the set of all **derivable** components, i.e., for every category c that can appear in a sentential form of a derivation in some CCG, G, each component of c is in $D_C(G)$. This can be shown, since, for each combinatory rule, if it holds of the categories on the left of the rule then it will hold of the category on the right. The number of derivable *categories* is unbounded because they can have an unbounded number of immediate components.

Each of the combinatory rules in a CCG can be viewed as a statement about how a pair of categories can be combined. For the sake of this discussion, let us name the members of the pair according to their role in the rule.

> The first of the pair in forward rules and the second of the pair in backward rules will be named the **principal** category. The second of the pair in forward rules and the first of the pair in backward rules will be named the **subordinate** category.

As a result of the form that combinatory rules can take in a CCG, they have the following property. When a combinatory rule is used, there is a bound on the number of immediate components that the subordinate categories of that rule may have. Thus, because immediate components must belong to $D_C(G)$ (a finite set), there is a bound on the number of categories that can fill the role of subordinate categories in the use of a combinatory rule. Thus, there is a bound on the number of instantiations of each of the variables y and z_i in the combinatory rules in Section 5.1. The only variable that can be instantiated to an unbounded number of categories is

x. Thus, we can create instances of the schematic combinatory rules by instantiating the object level variables y and each z_i with every possible combination of the finite number of bindings that they can take. Hence, the number of combinatory rules in R (while remaining finite) can be increased in such a way that only x is needed. Notice that x will appears only once on each side of the rules (i.e., they are linear).

We are now in a position to describe how to represent each of the combinatory rules by a production in the LIG, G'. In the combinatory rules, categories can be viewed as stacks since symbols need only be added and removed from the right. The subordinate category of each rule will be a ground category: either A, or $(A|_1c_1|_2\ldots|_nc_n)$, for some $n \geq 1$. These can be represented in a LIG as $A[\,]$ or $A[|_1c_1|_2\ldots|_nc_n]$, respectively. The principal category in a combinatory rule will be unspecified except for the identity of its left and rightmost immediate components. If its leftmost component is a nonterminal, A, and its rightmost component is a member of $D_C(G)$, c, this can be represented in a LIG by $A[\cdot\cdot c]$.

In addition to mapping combinatory rules onto productions we must include productions in G' for the mappings from lexical items.

If $A \in f(a)$ where $a \in V_T \cup \{\epsilon\}$ then $A[\,] \rightarrow a \in P$

If $(A|_1c_1|_2\ldots|_nc_n) \in f(a)$ where $a \in V_T \cup \{\epsilon\}$ then $A[|_1c_1|_2\ldots|_nc_n] \rightarrow a \in P$

We now illustrate this construction by giving a LIG from a CCG that generates the language

$$\{\, a^nb^nc^nd^n \mid n \geq 0 \,\}$$

Example 5.2.1 Let a CCG be defined as follows, where we have omitted unnecessary parenthesis.

$$A \in f(a) \qquad B \in f(b) \qquad C \in f(c) \qquad D \in f(d)$$

$$S/S_1 \in f(\epsilon) \qquad S_1 \in f(\epsilon) \qquad S_1\backslash A/D/S_1\backslash B/C \in f(\epsilon)$$

The following combinatory rules are permitted.

- Forward application involving x/y and y either when x begins with S_1 and y is C, or when x begins with S and y is S_1 or D.
- Backward application involving y and $x\backslash y$ when x begins with S and y is A or B.
- Forward composition involving x/y and $y\backslash z_1/z_2/z_3\backslash z_4$ when x begins with S and y is S_1.

The productions of the LIG that would be constructed from this CCG are as follows. The first 7 rules result from the definition of f.

$$A[\,] \to a \qquad B[\,] \to b \qquad C[\,] \to c \qquad D[\,] \to d$$

$$S_1[\,] \to \epsilon \qquad S_1[\backslash A/D/S_1\backslash B/C] \to \epsilon \qquad S_1[\cdot\cdot] \to S_1[\cdot\cdot/C]\, C[\,]$$

$$S[/S_1] \to \epsilon \qquad S[\cdot\cdot] \to S[\cdot\cdot/S_1]\, S_1[\,] \qquad S[\cdot\cdot] \to S[\cdot\cdot/D]\, D[\,]$$

$$S[\cdot\cdot] \to A[\,]\, S[\cdot\cdot\backslash A] \qquad S[\cdot\cdot] \to B[\,]\, S[\cdot\cdot\backslash B]$$

$$S[\cdot\cdot\backslash X_1/X_2/X_3\backslash X_4] \to S[\cdot\cdot/S_1]\, S_1[\backslash X_1/X_2/X_3\backslash X_4]$$

for all $X_1, \ldots, X_4 \in V_N$.

5.2.2 TAL's $\subseteq$ CCL's

We briefly describe the construction of a CCG, G' from a TAG, G, such that G and G' are equivalent. In order to understand this construction, it is important to appreciate that the order in which categories are combined is crucial in a CCG derivation[2]. We must therefore take care to ensure that a particular derivation order will occur. We will assume that the TAG is in a normal form corresponding to that arising from the construction from HG's to TAG's, i.e., there is one initial tree, and 5 types of auxiliary trees.

Each of the auxiliary trees will result in certain assignments of categories by f to a terminal or the empty string. Each occurrence of adjunction will be mimicked by the use of a combinatory rule.

Adjunction into nodes to the right (left) of the foot node (which corresponds to concatenation) will be simulated by backward (forward) application. Adjunction into nodes dominating the foot node of a tree (which corresponds to wrapping) will be simulated in the CCG by composition. It is necessary that we ensure that the subsidiary category in every occurrence of composition has just been introduced into the derivation by an assignment of f (see Figure 5.1). This will correspond to the adjunction of an auxiliary tree that has not had any trees adjoined into it.

We can guarantee that composition only occurs in this context in the following way. For each nonterminal, A of G there will be two nonterminals A^a and A^c in G'. We restrict each combinatry rule so that the subsidiary category of the rule has the form A^a (for application) or $(A^c|_1c_1 \ldots |_nc_n)$ (for composition). The the other categories in the rule has as its target category some B^a.

[2]This is discussed in Section 5.6.

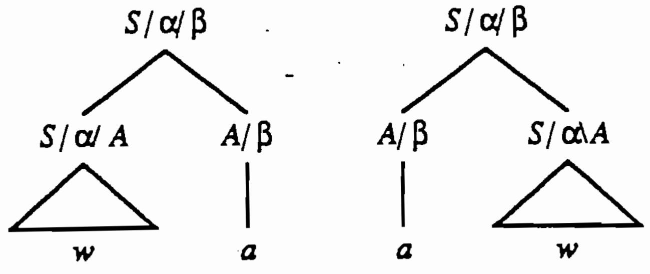


Figure 5.1: Contexts of composition

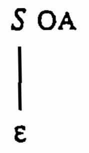
Forward and backward application are restricted to cases where the subordinate category is some X^a, and the left immediate component of the principal category is some Y^a.

Forward and backward composition are restricted to cases where the subordinate category has the form $((X^c|_1c_1)|_2c_2)$, or $(X^c|_1c_1)$, and the left immediate component of the principal category is some Y^a.

An effect of the restrictions on the use of combinatory rules is that only categories that can fill the subordinate role during composition are categories assigned to terminals by f. Notice that the combinatory rules of G' depend only on the terminal and nonterminal alphabet of the TAG, and are independent of the elementary trees.

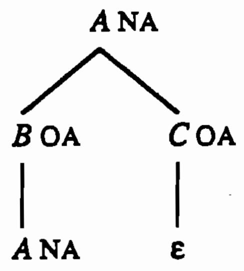
f is defined on the basis of the auxiliary trees in G. Without loss of generality we assume that the TAG, G, has trees of the following form (see [87, 89, 43]).

I contains one initial tree



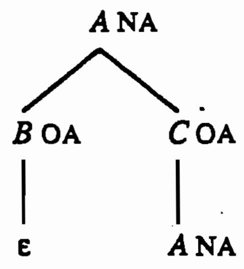
Thus, in considering the language derived by G, we need only be concerned with trees derived from auxiliary trees whose root and foot are labeled by S. There are 5 kinds of auxiliary trees in A.

1. For each tree of the form



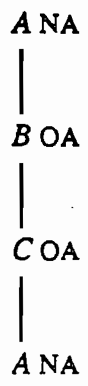
include $A^a/C^a/B^c \in f(\epsilon)$ and $A^c/C^a/B^c \in f(\epsilon)$

2. For each tree of the form



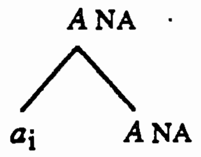
include $A^a\backslash B^a/C^c \in f(\epsilon)$ and $A^c\backslash B^a/C^c \in f(\epsilon)$

3. For each tree of the form



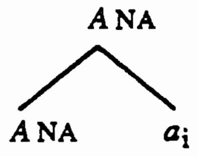
include $A^a/B^c/C^c \in f(\epsilon)$ and $A^c/B^c/C^c \in f(\epsilon)$

4. For each tree of the form



include $A^a\backslash A_i \in f(\epsilon)$, $A^c\backslash A_i \in f(\epsilon)$ and $A_i \in f(a_i)$

5. For each tree of the form include



include $A^a/A_i \in f(\epsilon)$, $A^c/A_i \in f(\epsilon)$ and $A_i \in f(a_i)$

The start symbol of G' is S^a. The CCG, G', in deriving a string, can be understood as mimicking a derivation in G of that string in which trees are adjoined in a particular order, that we now describe. We define this order by describing the set, $T_i(G)$, of all trees produced in i or fewer steps, for $i \geq 0$.

$T_0(G)$ is the set of auxiliary trees of G.

$T_i(G)$ is the union of $T_{i-1}(G)$ with the set of all trees γ produced in one of the following two ways.

1. Let γ' and γ'' be trees in $T_{i-1}(G)$ such that there is a unique lowest OA node, η, in γ' that does not dominate the foot node, and γ'' has no OA nodes. γ is produced by adjoining γ'' at η in γ'.

2. Let γ' be a tree in $T_{i-1}(G)$ such that there is OA node, η, in γ' that dominates the foot node and has no lower OA nodes. γ is produced by adjoining an auxiliary tree β at η in γ'.

Each tree $\gamma \in T_i(G)$ with frontier w_1Aw_2 has the property that it has a single spine from the root to a node that dominates the entire string w_1Aw_2. All of the OA nodes remaining in the tree fall on this spine, or hang immediately to its right or left. For each such tree γ there will be a derivation tree in G', whose root is labeled by a category c and with frontier w_1w_2, where c *encodes* the remaining obligatory adjunctions on this spine in γ.

We define how a category c encodes the spine as follows.

- If the spine contains no OA nodes then $c = A^a$ where A labels the root of the tree.
- If there is a unique lowest OA node labeled by B that falls on the left (resp. right) of the spine, and c' encodes the remainder of the spine, then the entire spine is encoded by $c'\backslash B^a$ (resp. c'/B^a).
- If the lowest OA node on the spine is as low as any other OA node, is labeled by B, and c' encodes the remainder of the spine, then the entire spine is encoded by c'/B^c. (in this case the direction of the slash was arbitrarily chosen to be forward)

For example, the tree shown in Figure 5.2 is encoded by the category

$$A\backslash A_1^a/A_2^c/A_3^a\backslash A_4^a$$

We show that $L(G) = L(G')$ by proving the following lemma.

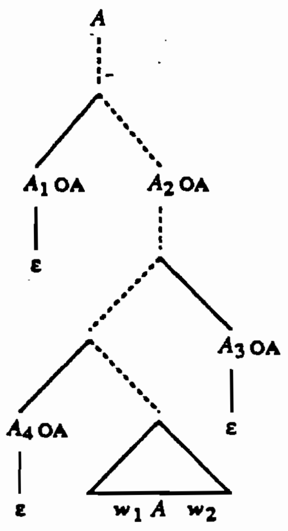


Figure 5.2: Tree encoding $A\backslash A_1^a/A_2^c/A_3^a\backslash A_4^a$

Lemma 5.2.1 $\gamma \in T_i(G)$ with frontier w_1Aw_2 for some $i \geq 0$ where the category c encodes the spine of γ if and only if c derives the string w_1w_2 in G' and the last category in c has the form A^a or A^c for some $A \in V_N$.

Proof: We first prove the *only if* direction by induction on i. For $i = 0$ we must consider the auxiliary trees of G. If follows directly from the construction that the spine of each auxiliary tree is encoded by a category that derives the string on the frontier of the tree. Consider a tree $\gamma \in T_i(G)$ for some $i \geq 1$. γ may have been obtained in a number of ways.

1. γ is produced by adjoining γ'' at η in γ', where η is on the left of the spine. Since γ' and γ'' are trees in $T_{i-1}(G)$, by induction, there are categories c' and c'' encoding their respective spines and deriving their respective frontier terminal strings (w_1 and w_2). Since γ' has

no OA nodes, $c' = B^a$ where b labels η. c'' must be of the form $c\backslash B^a$, so by backward application, we can obtain c deriving the string $w_1 w_2$; where c encodes the spine of γ and γ derives $w_1 w_2$.

2. There is a second case in which adjunction takes place to the right of the spine. It is analogous to the first case.

3. The final case involves adjunction on the spine. Let γ be a tree in $T_i(G)$ produced by adjoining an auxiliary tree β at η in $\gamma' \in T_{i-1}(G)$ where η is labeled by B and dominates the foot node. By induction, there will be a category $c' = c/B^c$ deriving w, encoding the spine and frontier of γ'. β may be any of the five types of auxiliary trees in G whose root and foot is labeled by B. For each type of the first three types of auxiliary trees there will be a category beginning with B^c in $f(\epsilon)$ that can be combined with c' using forward composition to give a category encoding the spine of γ. In the trees whose frontier may contain terminals, again there will be a category beginning with B^c in $f(\epsilon)$ that can be combined with c' using forward composition. We then use either forward or backward application to introduce $a_i \in V_T \cup \{\epsilon\}$ on the correct side of the spine.

We prove the *if* direction of the lemma by induction on the number of combinatory rules that are used. The basis involves consideration of categories in the range of f. It follows from the construction that the lemma will hold of these categories.

The category c is obtained by combining two categories c_1 and c_2 using one of the combinatory rules. By induction we assume that c_1 and c_2

derive w_1 and w_2, and are encoded by γ_1 and γ_2, respectively. Suppose, c_1 and c_2 are combined using forward application. $c_2 = A^a$ and γ_2 is an auxiliary tree with root labeled by A and no OA nodes. $c_1 = c/A^a$ and γ_1 has its lowest OA node labeled by A to the right of the spine. Thus γ_2 can be adjoined at this node in γ_1 to produce a tree γ with frontier w_1w_2 that is encoded by c. The case of backward application is analogous. Let us consider forward composition. The category c_2 begins with some category A^c, therefore c_2 must be a category in the range of f, since it can not result from the use of a combinatory rule. Thus, γ_2 is an auxiliary tree corresponding to c_2 whose root and foot are labeled by A. c_1 must be of the form c'/A^c, thus γ_1 contains an OA node dominating the foot labeled A at which γ_2 can be adjoined to produce γ. The category resulting from composing c_1 and c_2 may end with a category A_i, in which case application of c and A_i is used, immediately to produce c. In the other cases, c results directly from the composition of c_1 and c_2. By consideration of each type of auxiliary tree it can be seen that γ will be encoded by c. $\square$

Example 5.2.2 Figure 5.3 shows an example of a TAG for the language $L_2 = \{\, a^n b^n \mid n \geq 0\}$ with crossing dependencies. We give the CCG that would be produced according to this construction.

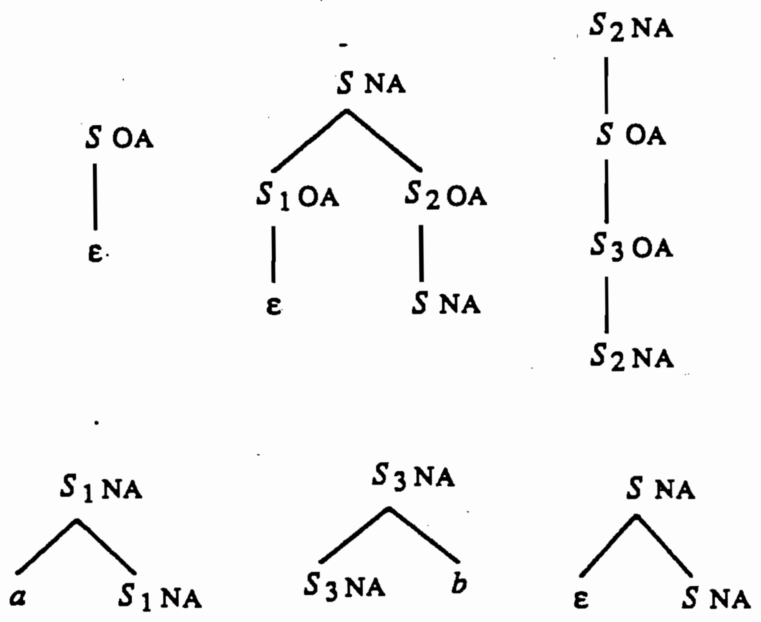


Figure 5.3: TAG for L_2

$S^a\backslash S_1^a/S_2^c \in f(\epsilon)$ $\quad$ $S^c\backslash S_1^a/S_2^c \in f(\epsilon)$

$S_2^a/S^c/S_3^c \in f(\epsilon)$ $\quad$ $S_2^c/S^c/S_3^c \in f(\epsilon)$

$S_1^a\backslash A \in f(\epsilon)$ $\quad$ $S_1^c\backslash A \in f(\epsilon)$

$S_3^a/B \in f(\epsilon)$ $\quad$ $S_3^c/B \in f(\epsilon)$

$A \in f(a)$ $\quad$ $B \in f(b)$

$S^a\backslash S_\epsilon \in f(\epsilon)$ $\quad$ $S^c\backslash S_\epsilon \in f(\epsilon)$

$S_\epsilon \in f(\epsilon)$

The CCG's produced according the construction given here have the property that parenthesis always have leftmost association, and can therefore be omitted. Thus, it is a corollary of our proofs in this section that the use of parenthesis in CCG's does not increase the generative power of the formalism. Note that the

Parenthesis-free Categorial Grammars of [23, 22] differ from CCG's as described here. We show below that they generate languages that can be be produced by any CCG.

5.3 Increasing the Power of CCG's

In this section we discuss some additions that have been used in certain papers on CCG's that add to the power. These were used because they were seen as being useful in analyzing certain coordination phenomenon in Dutch. We show that they increase the power of CCG's suggesting that they should perhaps be avoided unless this extra power is actually needed.

5.4 Schema for Coordination

A characteristic feature of LCFRS's is that they are unable to produce two structures exhibiting an unbounded dependence. It has been suggested that this capability may be needed in the analysis of coordination in Dutch, and an extension of CCG's has been proposed by Steedman [78] in which this is possible. The following schema is included.

$$x^+ \ \mathrm{conj}\ x \rightarrow x$$

where, in the analysis given of Dutch, x is allowed to match categories of arbitrary size. Two arbitrarily large structures can be encoded with two arbitrarily large categories. This schema has the effect of checking that the encodings are identical. The addition of rules such as this increases the generative power of CCG's, e.g., the following language can be generated.

$$\{(wc)^n \mid w \in \{a,b\}^*\}$$

We believe that this language can not be generated by an LCFRS[3]. In giving analysis of coordination (in languages other than Dutch), only a finite number of instances of this schema are required since only bounded categories can be coordinated. This form of coordination can thus be encapsulated in the assignments to lexical items, and does not require the use of powerful coordination schema.

In addition to the schema given above, Steedman [78] considers type raising schema. These schema can be applied to the categories assigned to lexical items to produce arbitrarily many categories. An example of such a schema follows.

$$B \rightarrow ((A|\alpha)/((A|\alpha)\backslash B))$$

This indicates that if the category B is assigned to a lexical item then for any possible category $(A|\alpha)$ a type raised category $((A|\alpha)/((A|\alpha)\backslash B))$ can be assigned to the lexical item. With the addition of such schema, the proofs used in the earlier section to show the equivalence of CCG's with TAG's would not go through.

5.5 Generalized Composition

Steedman [78] considers a CCG in which there may be an *infinite* number of composition rules. For each $n \geq 1$ there can be a rule of the form

$$(x/y) \quad (\ldots(y|_1 z_1)|_2 \ldots |_n z_n) \rightarrow (\ldots(x|_1 z_1)|_2 \ldots |_n z_n)$$

$$(\ldots(y|_1 z_1)|_2 \ldots |_n z_n) \quad (x\backslash y) \rightarrow (\ldots(x|_1 z_1)|_2 \ldots |_n z_n)$$

[3]Although the details of how these schema can be used in a grammar have not been formalized, it appear that in their general form they would make CCG's equivalent to IG's

With this addition, the generative power of CCG's increases. We show this by giving a grammar for a language that is known not to be a Tree Adjoining-language. Consider the following CCG. We allow unrestricted use of *arbitrarily* many combinatory rules for forward or backwards generalized composition and application.

$$\begin{array}{ll} f(\epsilon) = \{S\} & \\ f(a_1) = \{A_1\} & f(b_1) = \{B_1\} \\ f(a_2) = \{A_2\} & f(b_2) = \{B_2\} \\ f(c_1) = \{S\backslash A_1/D_1/S\backslash B_1\} & f(d_1) = \{D_1\} \\ f(c_2) = \{S\backslash A_2/D_2/S\backslash B_2\} & f(d_2) = \{D_2\} \end{array}$$

When the language L, generated by this grammar is intersected with the regular language $\{ a_1^* a_2^* b_1^* c_1^* b_2^* c_2^* d_2^* d_1^* \}$ we get the following language.

$$L' = \{ a_1^{n_1} a_2^{n_2} b_1^{n_1} c_1^{n_1} b_2^{n_2} c_2^{n_2} d_2^{n_2} d_1^{n_1} \mid n_1, n_2 \geq 0 \}$$

The pumping lemma for Tree Adjoining Grammars [82] can be used to show that L' is not a Tree Adjoining Language. The pumping lemma can be stated as follows.

Theorem 5.5.1 If L is a TAL, then there is a constant n such that if $z \in L$ and $|z| \geq n$ then

1. $z = u_1 v_1 w_1 v_2 u_2 v_3 w_2 v_4 u_3$
2. $|v_1 w_1 v_2 v_3 w_2 v_4| \leq n$
3. $|v_1 v_2 v_3 v_4| \geq 1$
4. $u_1 v_1^i w_1 v_2^i u_2 v_3^i w_2 v_4^i u_3 \in L$ for all $i \geq 0$

This pumping lemma states that we must allow pumping at 4 points in the string. It also states that there is a bound on the length of w_1 and w_2. Although there are 4 positions that when pumped would continue to yield strings in L', we can not find 4 suitable places to pump such that the gaps corresponding to w_1 and w_2 are bounded. Hence L' can not be a TAL. Since L' is not a Tree Adjoining Language and Tree Adjoining Languages are closed under intersection with Regular Languages, L can not be a Tree Adjoining Language either.

This form of composition is permitted in the Parenthesis-free Categorial Grammars which have been studied in [23, 22]. Thus, a consequence of the example given in this section is that Parenthesis-free Categorial Grammars are more powerful than CCG's.

In the remainder of this section we briefly consider changes to LIG's and NPDA's that appear to directly correspond to the addition of these schematic composition rules to CCG's. We make no attempt to prove that these extensions to CCG's, LIG's, and the NPDA produce equivalent systems.

The appropriate addition to LIG's would appear to require rules of the form

$$A[\alpha_1\alpha_2] \rightarrow B[\alpha_1]C[\alpha_2]$$

in which the unbounded stack $[\alpha_1\alpha_2]$ has an *unbounded* number of symbols α_2 popped from it.

Example 5.5.1 The language L' given above can be generated by the following extended LIG.

$$\begin{array}{ll}
S[\cdot\cdot] \rightarrow a_1S[\cdot\cdot l_1]d_1 & S[\cdot\cdot] \rightarrow a_2S[\cdot\cdot l_2]d_2 \\
S[\alpha_1\alpha_2] \rightarrow S_1[\alpha_1]S_2[\alpha_2]S_1[\cdot\cdot l_1] \rightarrow b_1S_1[\cdot\cdot]c_1 & S_1[\,] \rightarrow \epsilon \\
S_2[\cdot\cdot l_2] \rightarrow b_2S_2[\cdot\cdot]c_2 & S_2[\,] \rightarrow \epsilon
\end{array}$$

A similar extension can be made to the NPDA's. An unbounded number of symbols can be popped off the top stack by permitting the NPDA transition function to specify that the top stack should be split into two unboundedly large separate stacks.

5.6 Derivations Trees

We examine derivation trees of CCG's and compare them to those of LCFRS's. In order to compare CCG's with other systems we must choose a suitable method for the representation of derivations in a CCG. In the case of CFG, TAG, HG, for example, it is fairly clear what the elementary structures and composition operations should be, and as a result, in the case of these formalisms, it is apparent how to represent derivations.

The traditional way in which derivations of a CCG have been represented has involved a binary tree whose nodes are labeled by categories with annotations indicating which combinatory rule was used at each stage. These derivation trees are different from those systems in the class of LCFRS's in two ways. They have context-free path sets, and the set of categories labeling nodes may be infinite. A property that they share with LCFRS's is that there is no dependence between unbounded paths. In fact, the derivation trees sets produced by CCG's have the same properties as those produced by LIG's (this is apparent from the construction in Section 5.2.1).

Although the derivation trees that are traditionally associated with CCG's differ from those of LCFRS's, this does not preclude the possibility that there may be an alternative way of representing derivations. What appears to be needed is some characterization of CCG's that identifies a finite set of elementary structures and a finite set of composition operations.

Let us consider what is in many ways the most natural way of attempting to represent CCG derivations with a tree in which the function argument structure is given. When some lexical item a has the category $c = (A|_1 c_1 \cdots |_n c_n)$, we can think of a as having the category of a n argument function from objects of categories $c_1, \ldots, c_n$ to an object of category A, the target category. We can therefore use a tree to represent the function argument structure given by a CCG derivation of A using the assignment of c to the lexical item a. The tree would be rooted in a node labeled by A with n subtrees showing the function argument structure of the n arguments $c_1, \ldots, c_n$.

These function argument trees do not, however, encode the CCG derivation since it is not in general possible to recover the derivation from such a tree. The same function argument tree could result from distinct derivations of the same string. This has led to the suggestion that CCG's "suffer" from *spurious* ambiguity. However, derivations of *different* strings can map onto the same function argument tree. We show this with the following example. Suppose that we could produce the three CCG derivations shown in Figure 5.4. These three trees could be combined in two possible ways, (shown in Figure 5.5) producing the *same* function argument tree. Notice that the two trees in Figure 5.5 have distinct strings on their frontier. This shows that CCG's can not really be said to suffer from spurious ambiguity, but instead, they can make more fine grained distinctions than are present in a function argument tree[4].

It is possible that there is some way of adding a finite number of annotations to

[4]It is interesting to note that this example is not possible in an alternative version of Categorial Grammars considered by Moortgat [55] that derives from the Lambek Calculus discussed in [81]. In Moorgat's system, disharmonic composition (involving slashes of mixed directionality) of the kind used in this example is not permitted.

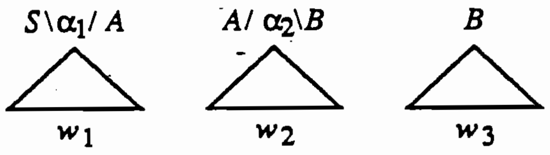


Figure 5.4: Three partial CCG derivations

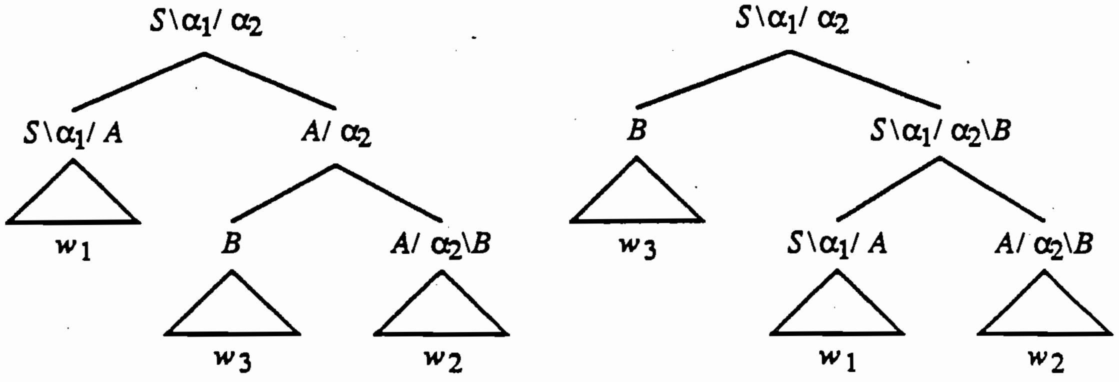


Figure 5.5: Alternative derivations

these function argument trees so that they are true derivation trees from which a unique derivation can be recovered.

The equivalence of TAG's and CCG's suggests one approach to this. The construction that we gave from TAG's to CCG's produced CCG's having a specific form which can be thought of as a normal form for CCG's. We can represent the derivations of grammars in this form with the same tree sets as the derivation tree sets of the TAG from which they were constructed. Hence CCG's in this normal form can be classified as LCFRS's.

TAG derivation trees encode the adjunction of specified elementary trees at specified nodes of other elementary trees. Thus, the nodes of the derivation trees are labeled by the names of elementary trees and tree addresses. In the construction used in Section 5.2.2, each auxiliary tree produces assignments of *elementary* categories to lexical items. CCG derivations can be represented with trees whose nodes identify elementary categories and specify which combinatory rule was used to combine it. For grammars in this normal form, a unique derivation can be recovered from these trees, though, as we have seen, this is not true of arbitrary CCG's where different orders of combination of the elementary categories can result in derivations that must be distinguished. In this normal form, the combinatory rules are so restrictive that there is only one order in which elementary categories can be combined. Without such restrictions, this style of derivation tree must encode the order of derivation.

Chapter 6

Conclusions

This dissertation has been concerned with formalizations of linguistically and computationally important aspects of grammar formalisms. Much of the earlier research in this area has concerned matters of weak generative capacity. Although we have also looked at this property of formalisms, and shown that various formalisms generate the same string languages, we have in addition, made an attempt to investigate other aspects of the descriptive capacity of formalisms. Most attention is given to formalisms with highly constrained generative power, in particular, a class of formalisms that has been named mildly context-sensitive grammar formalisms. This class is believed to contain formalisms that are suitable for the representation of natural language grammars.

In Chapter 2 we examined the tree sets or structural descriptions produced by several formalisms in terms of their path sets and the dependence between paths. We show that Tree Adjoining Grammars (TAG's) have path sets that, like those of Indexed Grammars (IG's), are context-free languages. We found that formalisms (such as IG's) that are able to compare unbounded hierarchical structures have trees sets in which have dependence between paths. Although the notion of dependence between paths is intuitively reasonably clear, we have not yet been able to give a suitable definition that applies to arbitrary tree sets. Formalisms such as IG's generated tree sets with unbounded number of unboundedly large paths. We also observed that tree manipulating formalisms such as TAG's had both object and meta level tree sets associated with them. The meta level or derivation tree sets of many of the formalisms that were considered in Chapter 2 where essentially identical. We take this to be a reflection of an underlying similarity between these notationally quite different formalisms. These derivation trees reflect the fact that in each case the formalism's derivation process is context-free. That is, choices about how to rewrite at each stage in the derivation are independent of the derivational context. Derivation trees prove to be useful objects to consider since they represent an abstraction away from the details of the formalism.

In Chapter 3 we extended the observations of Chapter 2 by describing a range of new formalisms that share many of the properties characterized in Chapter 2. Several recent results, including those of Chapter 5 have shown that there is a class of languages generated by a number of grammar formalisms that falls slightly outside Context-Free Languages (CFL's). For a number of reasons this appears to be a natural class of languages, and in a sense, is one step up from CFL's. We described a number of progressions that shed light on the relationship between this class and CFL's. These progressions are defined in terms of tree sets, grammar formalisms, automata, and generators. There are many questions that arise out of these definitions concerning the relationship between each of the progressions, and other formal properties, such as closure properties, of members of each class. We have answered some of these questions, but substantial amount of work remains before these progressions will be satisfactorily characterized. In addition, there are other characterizations of a similar progression that we have not given, for example, in terms of tree automata. We believe that in this chapter, an account is provided of the notion of "limited" crossed dependencies, as they appear in Tree Adjoining Languages (TAL's), Head Languages (HL's), Combinatory Categorial Languages (CCL's) and Linear Indexed Languages (LIL's). According to this analysis, crossed dependencies of this form arise from the simultaneous application of 2 nested dependencies.

Based on the observation that a number of grammar formalisms have similar derivation tree sets to those of Context-Free Grammars (CFG's), in Chapter 4 we attempt to formalize the class of all such formalisms, which we call Linear Context-Free Rewriting Systems (LCFRS's). Not only are restrictions placed on the derivation tree sets, but we restrict the composition operations to be linear and non-erasing. We then investigate the computational properties of these systems. Intuitively, there appeared to be a relationship between the notion of independence of paths and the ability to perform recognition based on efficient divide-and-conquer based recognition. Indeed based on a result in [67] we find that the recognition problem for the class of languages generated by such systems is properly contained in P (the class of polynomial-time solvable languages). In [67] a class of systems is given whose recognition problems are exactly the class P. Upper bounds for the complexity of recognition are given for various subclasses of these systems. These upper bounds are determined by the efficiency of the Alternating Turing Machines produced by the construction given in proving the inclusion in P. These appear to be very loose bounds since, for example, TAL's and HL's get an upper bound of $O(n^{20})$). It would be interesting to investigate whether, given the additional restrictions of LCFRS's. tighter upper bounds can be found. In Chapter 4, we show that each languages is semilinear, and hence constant growth. It might be possible to extend the proof of the pumping lemma for CFL's to give weak pumping lemmas each LCFRS's, based on certain parameters of the system.

We show that the class of languages generated by LCFRS's is equal to the class generated by the extension of TAG's in which sequences of trees are simultaneously adjoined in sequences of trees (Multicomponent TAG's or MCTAG's). From the proof of this result, it is clear that MCTAG's are equivalent to a similar generalization of CFG's (in which finite sequences of productions are used to simultaneously rewrite the nonterminals on the right-hand sides of another sequence of productions).

The restrictions placed on the composition operations of LCFRS's are given mainly in terms of their effect on the terminal strings. It would be preferable to find some general way of formalizing these restrictions in terms of the structures (e.g., trees, tree sequences, or directed acyclic graphs) that the formalism manipulates. The intuition that we would like to formalize is that the operations should add together their arguments without copying or deleting unbounded structure, and in a sense they should not "restructure" their arguments. We would like to have some general way of describing this restriction which can apply to arbitrary rewrite systems.

Chapter 5 investigates Combinatory Categorial Grammars (CCG's), and extension of Classical Categorial Grammars. We describe a formalization of these grammars and show that it is weakly equivalent to TAG's, Head Grammars (HG's) and Linear Indexed Grammars (LIG's). As a consequence of establishing their relationship with other grammar formalisms, the results of this section show that CCL's inherit the results that have previously been established for these other systems. In particular, this results shows that CCG's can be recognized in polynomial time (in fact an upper bound of $O(n^6)$ has been given). We also show that additions to the system that have been used for certain Dutch coordination phenomena significantly increase the power. In doing so, we also show that Parenthesis-Free Categorial Grammars have more power than CCG's. It is an open question whether these Dutch coordination phenomena can be described using a grammar formalism that has highly restricted generative power. The machinery that has been added to CCG's appears to give them the same weak generative power as IG's. In Chapter 5 we also considered various ways in which CCG derivations can be represented. For reasons that are explained in Chapter 5, it is difficult to emulate the approach adopted to represent the derivation trees of a TAG. It was therefore not possible to show that CCG's possessed the necessary restrictions for membership of the class of LCFRS's. Lambek Calculus as studied in [55] is a formalism that is closely related to CCG's, but whose generative capacity has not been related to CFG's. It would be interesting to investigate whether this system is a LCFRS.

In this dissertation we have compared several of grammar formalisms, in particular those that have been described as mildly context-sensitive. We have identified a number of important common properties of some of these formalisms, as well as ways in which they differ. Perhaps the three most notable findings arising from our work are the following. (1) We found four formalisms (TAG's, HG's, CCG's, and LIG's) whose weak generative power were identical. (2) We established the relationship between this class of languages and CFL's by identifying a natural progression from regular languages to context-free languages to the TAL's (HL's, CCL's and LIL's) and beyond. (3) We found that a number of formalisms, having a range of weak generative power could all be viewed has having the same underlying context-free derivation process.

Bibliography

[1] S. Abraham. Some questions of phrase structure grammars. *Comput. Ling.*, 4:61–70, 1965.

[2] A. E. Ades and M. J. Steedman. On the order of words. *Ling. and Philosophy*, 3:517–558, 1982.

[3] A. V. Aho. Indexed grammars — An extension to context free grammars. *J. ACM*, 15:647–671, 1968.

[4] A. V. Aho. Nested stack automata. *J. ACM*, 16:383–406, 1969.

[5] K. Ajdukiewicz. Die syntaktische konnexitatat. *Studia Philosophica*, 1:1–27, 1935. English translation in: Polish logic 1920-1939, ed. by Storrs McCall, 207-231. Oxford University Press.

[6] E. Altman and R. Banerji. Some problems of finite representability. *Inf. Control*, 8:251–263, 1965.

[7] Y. Bar-Hillel, C. Gaifman, and E. Shamir. On categorial and phrase structure grammars. In *Language and Information*, Addison-Wesley, Reading, MA, 1964.

[8] G. E. Barton, R. Berwick, and E. S. Ristad. *Computational Complexity and Natural Language*. MIT Press, Cambridge, MA, 1987.

[9] R Berwick. Computational complexity and lexical-functional grammar. *American J. of Comput. Ling.*, 8(3-4):97–109, 1982.

[10] R. Berwick. Strong generative capacity, weak generative capacity, and modern linguistic theories. *Comput. Ling.*, 10:189–202, 1984.

[11] R. Berwick and A. Weinberg. *The Grammatical Basis of Linguistic Performance*. MIT Press, Cambridge, MA, 1984.

[12] J. W. Bresnan, R. M. Kaplan, P. S. Peters, and A. Zaenen. Cross-serial dependencies in Dutch. *Ling. Inquiry*, 13:613–635, 1982.

[13] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28:114–122, 1981.

[14] N. Chomsky. Context-free grammars and pushdown storage. In *Quarterly Prog. Rept. No. 65*, pages 187–194, MIT Res. Lab Elect., Cambridge, MA., 1962.

[15] N. Chomsky. A note on phrase-structure grammars. *Inf. Control*, 2(2):393–395, 1959.

[16] N. Chomsky. On certain formal properties of grammars. *Inf. Control*, 2(2):137–167, 1959.

[17] N. Chomsky. *Syntactic Structures*. Mouton and Co., The Hague, 1964.

[18] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, IT2:113–124, 1956.

[19] N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, North-Holland Publishing Co., Amsterdam, 1963.

[20] C. Culy. The complexity of the vocabulary of bambara. *Ling. and Philosophy*, 8:345–351, 1985.

[21] J. Friant. *Grammaires ordonnes - grammaires matricielles.* Technical Report MA-101, Univ. de Montreal, 1968.

[22] J. Friedman, D. Dai, and W. Wang. The weak generative capacity of parenthesis-free categorial grammars. In 11th *Intern. Conf. on Comput. Ling.*, 1986.

[23] J. Friedman and R. Venkatesan. Categorial and Non-Categorial languages. In 24th *meeting Assoc. Comput. Ling.*, 1986.

[24] G. Gazdar. *Applicability of Indexed Grammars to Natural Languages.* Technical Report CSLI-85-34, Center for Study of Language and Information, 1985.

[25] G. Gazdar. Phrase structure grammars. In P. Jacobson and G. Pullum, editors, *The Nature of Syntactic Recognition*, D. Reidel, Dordrecht, Holland, 1982.

[26] G. Gazdar, E. Klein, G. K. Pullum, and I. A. Sag. *Generalized Phrase Structure Grammars.* Blackwell Publishing, Oxford, 1985. Also published by Harvard University Press, Cambridge, MA.

[27] S. Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages.* North Holland Publishing Co., Amsterdam, 1975.

[28] S. Ginsburg and S. A. Greibach. Abstract families of languages. *Mem. Am. Math. Soc.*, 87(1):1–32, 1969.

[29] S. Ginsburg, S. A. Greibach, and M. Harrison. One-way stack automata. *J. ACM*, 14:389–418, 1967.

[30] S. Ginsburg and E. H. Spanier. Control sets on grammars. *Math. Syst. Theory*, 2:159–177, 1968.

[31] S. Gorn. Processors for infinite codes of Shannon-Fano type. In *Symp. Math. Theory of Automata*, 1962.

[32] I. Guessarian. On pushdown tree automata. In *Proceedings of* 6th *CAAP*, 1981. Lecture Notes in Computer Science, Springer Verlag, Berlin.

[33] T. Hayashi. On derivation trees of indexed grammars — An extension of the *uvwxy*-theorem. Publication of the Research Institute for Mathematical Sciences, Kyoto University, 1973.

[34] C. Heycock. *The Structure of the Japanese Causitive.* Technical Report MS-CIS-87-55, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, 1987.

[35] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[36] A. K. Joshi. How much context-sensitivity is necessary for characterizing structural descriptions — Tree Adjoining Grammars. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Processing — Theoretical, Computational and Psychological Perspective*, Cambridge University Press, New York, NY, 1985. Originally presented in 1983.

[37] A. K. Joshi. How much hierarchical structure is necessary for sentence description? In 3rd *International Conference on Comput. Ling.*, 1971.

[38] A. K. Joshi. An introduction to tree adjoining grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*, John Benjamins, Amsterdam, 1987.

[39] A. K. Joshi. The relevance of tree adjoining grammars to generation. 1986. Presented at the *3rd International Workshop on Natural Language Generation*, Nijmegen, The Netherlands.

[40] A. K. Joshi. Word-order variation in natural language generation. In *AAAI-87*, 1987.

[41] A. K. Joshi and L. S. Levy. Constraints on local descriptions: Local transformations. *SIAM J. Comput.*, 1977.

[42] A. K. Joshi, L. S. Levy, and M. Takahashi. Tree adjunct grammars. *J. Comput. Syst. Sci.*, 10(1), 1975.

[43] A. K. Joshi, K. Vijay-Shanker, and D. J. Weir. *Tree Adjoining Grammars and Head Grammars.* Technical Report MS-CIS-86-1, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, 1986.

[44] R. Kaplan and J. Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge MA, 1983.

[45] M. Kay. Functional grammar. In 5th *Annual Meeting of the Berkeley Linguistics Society*, 1979.

[46] N. A. Khabbaz. *Generalized Context-Free Languages.* PhD thesis, The University of Iowa, 1972.

[47] N. A. Khabbaz. A geometric hierarchy of languages. *J. Comput. Syst. Sci.*, 8:142–157, 1974.

[48] A. S. Kroch. Asymmetries in long distance extraction in a tag grammar. In M. Baltin and A. S. Kroch, editors, *New Conceptions of Phrase Structure*, University of Chicago, Press, Chicago, IL, 1986.

[49] A. S. Kroch. Subjacency in a tree adjoining grammar. In A. Manaster-Ramer, editor, *Mathematics of Language*, John Benjamins, Amsterdam, 1987.

[50] A. S. Kroch and A. K. Joshi. Analyzing extraposition in a tree adjoining grammar. In G. Huck and A. Ojeda, editors, *Syntax and Semantics: Discontinuous Constituents*, Academic Press, New York, NY, 1986.

[51] A. S. Kroch and A. K. Joshi. *Linguistic Relevance of Tree Adjoining Grammars.* Technical Report MS-CIS-85-18, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, 1985.

[52] A. S. Kroch and B. Santorini. The derived constituent structure of West Germanic verb-raising construction. In *Proceedings of the Princeton Workshop on Grammar*, 1987.

[53] A. Lindenmayer. Mathematical models for cellular interactions in development, parts i-ii. *J. Theoret. Biol.*, 18:280–315, 1968.

[54] T. S. E. Maibaum. A generalized approach to formal languages. *J. Comput. Syst. Sci.*, 8:409–439, 1974.

[55] M. Moortgat. Lambek theorem proving. Presented at the ZWO Workshop. *Categorial Grammar: Its Current State.*, 1987.

[56] W. F. Ogden. *Intercalation Theorems for Pushdown Store and Stack Languages.* PhD thesis, Stanford University, 1968.

[57] M. Palis and S. Shende. *Upper Bounds on Recognition of a Hierarchy of Non-Context-Free Languages.* Technical Report MS-CIS-88-56, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, 1988.

[58] M. Palis, S. Shende, and D. S. L. Wei. An optimal linear-time parallel parser for tree-adjoining languages. 1986. Manuscript.

[59] R. Pareschi and M. J. Steedman. A lazy way to chart-parse with categorial grammars. In 25th *meeting Assoc. Comput. Ling.*, 1987.

[60] R. Parikh. On context free languages. *J. ACM*, 13:570–581, 1966.

[61] F. C. N. Pereira and D. Warren. Definite clause grammars for language analysis: A survey of the formalism and a comparison with augmented transition networks. *Artif. Intell.*, 13:231–278, 1980.

[62] C. Pollard. *Generalized Phrase Structure Grammars, Head Grammars and Natural Language.* PhD thesis, Stanford University, 1984.

[63] C. Pollard. Lecture notes on head-driven phrase-structure grammar. 1985. Center for the Study of Language and Information, Stanford University, Stanford, CA.

[64] G. Pullum and G. Gazdar. Natural languages and context-free languages. *Ling. and Philosophy*, 4:471–504, 1982.

[65] K. Roach. Formal properties of head grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*, John Benjamins, Amsterdam, 1987.

[66] W. C. Rounds. Context-free grammars on trees. In *ACM Symposium on Theory of Computing*, 1969.

[67] W. C. Rounds. LFP: A logic for linguistic descriptions and an analysis of its complexity. To appear in *Comput. Ling.*

[68] A. Salomaa. *Formal Languages.* Academic Press, New York, NY, 1973.

[69] B. Santorini. *The West Germanic Verb-Raising Construction: A Tree Adjoining Grammar Analysis.* Master's thesis, University of Pennsylvania, Philadelphia, PA, 1986.

[70] Y. Schabes, A. Abeille, and A. K. Joshi. Parsing strategies with 'lexicalized' grammars: applications to tree adjoining grammars. In 12th *International Conference on Comput. Ling.*, 1988.

[71] Y. Schabes and A. K. Joshi. An Earley-type parsing algorithm for tree adjoining grammars. In 26th *meeting Assoc. Comput. Ling.*, 1988.

[72] K. M. Schimpf and J. H. Gallier. Tree pushdown automata. *J. Comput. Syst. Sci.*, 30:25–39, 1985.

[73] S. Shende. Dissertation proposal. 1988. Dept. of Computer and Information Science, University of Pennsylvania.

[74] S. M. Shieber. Evidence against the context-freeness of natural language. *Ling. and Philosophy*, 8:333–343, 1985.

[75] M. Steedman. Combinators and grammars. In R. Oehrle, E. Bach, and D. Wheeler, editors, *Categorial Grammars and Natural Language Structures*, Foris, Dordrecht, 1986.

[76] M. Steedman. Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, 1987.

[77] M. Steedman. Gapping as constituent coordination. 1987. m.s. University of Edinburgh.

[78] M. J. Steedman. Dependency and coordination in the grammar of Dutch and English. *Language*, 61:523–568, 1985.

[79] J. W. Thatcher. Characterizing derivations trees of context free grammars through a generalization of finite automata theory. *J. Comput. Syst. Sci.*, 5:365–396, 1971.

[80] J. W. Thatcher. Tree automata: An informal survey. In A. V. Aho, editor, *Currents in the Theory of Computing*, pages 143–172, Prentice Hall Inc., Englewood Cliffs, NJ, 1973.

[81] J. van Benthem. Lambek calculus. 1985. Manuscript.

[82] K. Vijay-Shanker. *A Study of Tree Adjoining Grammars.* PhD thesis, University of Pennsylvania, Philadelphia, PA, 1987.

[83] K. Vijay-Shanker and A. K. Joshi. Feature based tree adjoining grammars. In 12th *International Conference on Comput. Ling.*, 1988.

[84] K. Vijay-Shanker and A. K. Joshi. Nested pushdown automata. In preperation.

[85] K. Vijay-Shanker and A. K. Joshi. Some computational properties of tree adjoining grammars. In 23rd *meeting Assoc. Comput. Ling.*, pages 82–93, 1985.

[86] K. Vijay-Shanker, D. J. Weir, and A. K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In 25th *meeting Assoc. Comput. Ling.*, 1987.

[87] K. Vijay-Shanker, D. J. Weir, and A. K. Joshi. Tree adjoining and head wrapping. In 11th *International Conference on Comput. Ling.*, 1986.

[88] D. J. Weir and A. K. Joshi. Combinatory categorial grammars: Generative power and relationship to linear context-free rewriting systems. In 26th *meeting Assoc. Comput. Ling.*, 1988.

[89] D. J. Weir, K. Vijay-Shanker, and A. K. Joshi. The relationship between tree adjoining grammars and head grammars. In 24th *meeting Assoc. Comput. Ling.*, 1986.

[90] K. B. Wittenburg. Natural language parsing with combinatory categorial grammar in a graph-unification based formalism. 1986. D.Phil. Dissertation, University of Texas at Austin.