A STUDY OF TREE ADJOINING GRAMMARS

K Vijayashanker

MS-CIS-88-03
LINC LAB 95

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

January 1988

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

A Study of Tree Adjoining Grammars

K Vijayashanker

A DISSERTATION

in

Computer and Information Sciences

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

1987

Supervisor of Dissertation

Graduate Group Chairperson

Acknowledgements

I would like to express my gratitude to my thesis supervisor, Professor Joshi, for his support and encouragement and for all the suggestions he gave me. I am extremely grateful for the numerous hours he spent guiding me through my thesis.

I would like to thank my thesis committee members, Professors J.H. Gallier, A. Kroch, M. Palis, and W. C. Rounds for their help and suggestions.

There are several friends who have helped me immeasurably during ny thesis. I am extremely grateful to David Weir for his help, for the countless hours we have spent discussing the work in this thesis and for all the suggestions he has made. I would also like to thank Atsushi Ohori and Vijay Gehlot for their suggestions following the discussions I have had with them.

I am extremely grateful to my grandfather, Dr. K. S. Ranganathan, who is a source of inspiration; to my wife, Shashi, for her encouragement and for always being there; to my parents, Krishnamurthi and Vatsala, who have supported me in all my endeavours; and to my brother, Raghu, and sister, Indu.

Abstract

A Study of Tree Adjoining Grammars

K Vijayashanker

Supervisor: A. K. Joshi

Constrained grammatical system have been the object of study in computational linguistics over the last few years, both with respect to their linguistic adequacy and their computational properties. A Tree Adjoining Grammar (TAG) is a tree rewriting system whose linguistic relevance has been extensively studied. A key property of these systems is that a TAG factors recursion from the co-occurence restrictions.

In this thesis, we study some mathematical properties of TAG's. We show that TAG's have several interesting properties and are a natural generalization of Context Free Grammars. We show the equivalence of the classes of languages generated by TAG's with those generated by Head Grammars and a linear version of Indexed Grammars, which have been studied for their linguistic applicability. We define the embedded pushdown automaton, an extention of the pushdown automaton, and prove that they are equivalent to TAG's. We show that the class of Tree Adjoining Languages form a substitution closed abstract family of languages, and that each Tree Adjoining Languiage is a semilinear language. We show that a TAG can be parsed in polynomial time by adapting the Cocke-Kasami-Younger algorithm for CFL's.

Feature structures, essentially a set of attribute value pairs, have been used in computational linguistics to to make statements of equality to capture some linguistic phenomena such as subcategorization and agreement. We embed TAG's in a feature structure based framework. We show that the resulting system has several advantages over TAG's. We give a mathematical model of this system based on the logical calculus developed by Rounds, Kasper, and Manaster-Ramer. Finally, we propose a restriction of this system and show how parsing of such a system can be done efficiently.

Contents

List of Figures

Chapter 1

Introduction

Constrained grammatical systems have been the object of study in computational linguistics, with repect to their linguistic adequacy as well as their computational properties. Tree Adjoining Grammars (TAG) is a constrained grammatical system which were first introduced by Joshi, Levy, and Takahashi [Joshi 75] and by Joshi [Joshi 85]. Tree Adjoining Grammars is a tree rewriting system that is slightly more powerful than Context Free Grammars in their generative capacity. This system has the feature of factoring recursion from the statement of co-occurence relations. Much of the linguistic relevance follows from this factorization. In this thesis, we study some mathematical properties of this grammatical system.

1.1 Constrained Grammatical Systems in Mathematical Linguistics

Constrained grammatical systems have been the object of study in mathematical linguistics. Some examples of constrained systems include Generalized Phrase Structure

(GPSG), Head Grammars (HG), Lexical Functional Grammars (LFG). Considerable research has been done in the formalization of structures arising in the analysis of natural languages, from the descriptive as well as the processing point of view. Although construction of such systems always involves some degree of abstraction, it is important that we do not depart from the structures actually arising in natural languages. Such research is important because it not only provides insights into the structure of natural languages, but it may also help in the design of efficient algorithms for natural language processing. A totally unconstrained system may not provide much insight into the structure of language.

Grammatical systems for natural language have two aspects, the mathematical formalism and the linguistic theory instantiated in that formalism. The latter component is a set of linguistic stipulations that place some further constraints on the set of structures derived by the formalism. If the grammatical formalism is too powerful, then the number of linguistic stipulations will increase. In constrained systems, a clearer distinction can be made between the linguistic constraints and the constraints enforced by the formalism itself. The number of linguistic stipulations could be reduced, as some linguistic stipulations could become corollaries of the constraints of the formalism itself and other linguistic stipulations. For example, in TAG's, the principle of subjacency is a necessary consequence of the formalism itself [Kroch 85]. Kroch [Kroch 87] notes that the empty category principle (ECP) receives a natural formulation in a TAG, and that the condition on extraction domains can be collapsed with ECP in a TAG.

1.2 Introduction to Tree Adjoining Grammars

Tree Adjoining Grammars (TAG), unlike other grammatical systems used in computational linguistics, is a tree rewriting system. Tree Adjoining Grammars were first introduced by Joshi, Levy, and Takahashi [Joshi 75]. The early study of this system, from the point of view of its properties and linguistic applicability was carried out by Joshi in [Joshi 85]. A more detailed study of the linguistic relevance of TAG's was done by Kroch and Joshi [Kroch 85]. Since then, many papers ([Kroch 86,Kroch 87,Santorini 86]) have appeared in which linguistic analysis of natural language constructions using TAG have been given.

Unlike the string rewriting formalisms which writes recursion into the rules that generate the phrase structure, a TAG factors recursion and dependencies into a finite set of elementary trees. A TAG is defined by a finite set of simple sentence *elementary trees* and uses an operation called *adjoining* to compose the trees. The elementary trees in a TAG correspond to *minimal* linguistic structures and localize the dependencies such as subcategorization and filler-gap. The unbounded distance between the dependent items in the surface constituent structure is a corollary of the operation of adjoining and the theory of sentential embedding. Thus, in a sense, there are no unbounded dependencies in a TAG. The TAG notation forces the embedded structures to be composed out of elementary structures that correspond to the minimal linguistic objects. The co-occurence relation between elements can be stated locally within the elementary trees. In this manner, the expression of recursion is partitioned from the co-occurence restrictions. Recursion is introduced by the use of adjoining.

A TAG is specified by a finite set of elementary trees. There are two kinds of elementary tree: the *initial trees* and *auxiliary trees*. The initial trees correspond to the structure for a simple sentence, one in which no recursive derivation occurs. Thus, the root of an initial tree is labelled by the symbol S. They are required to have a frontier

made up of terminal symbols. The structure of an initial tree, along with an example is given in Figure 1.1.

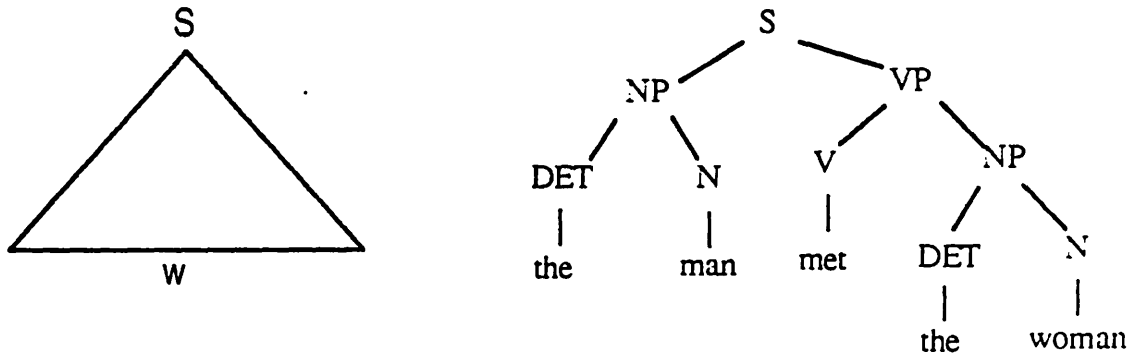

Figure 1.1: Initial Trees

The auxiliary trees correspond to recursion on a nonterminal. Therefore, if the root of an auxiliary tree is labelled by a nonterminal symbol, X, then there is a node in the frontier of this tree which is labelled by the same symbol X. The rest of the nodes in the frontier are labelled by terminal symbols. The node in the frontier which is labelled by the nonterminal symbol, X, is called the foot node of the auxiliary tree. Again, we expect the auxiliary tree to correspond to a minimal recursive structure that must be brought into the derivation when we recurse on the nonterminal X. In Figure 1.2, we give the structure of any auxiliary tree along with an linguistic example.

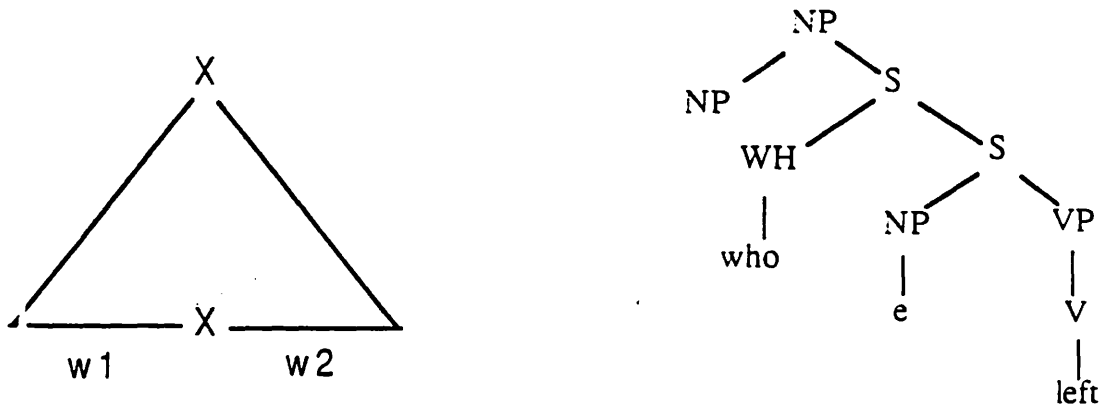

Figure 1.2: Auxiliary Trees

We will now define the operation of adjunction. Let γ be a tree with a node labelled by X. Let β be an auxiliary tree, whose root and foot node is labelled by X. Then, adjoining β at the node labelled by X in γ will result in the following tree. The tree below the node labelled by X in γ is excised, and the auxiliary tree β is inserted in its place. The excised subtree is then inserted below the foot node of β. This operation is illustrated in Figure 1.3. In Figure 1.4, we show the result of adjoining the auxiliary tree

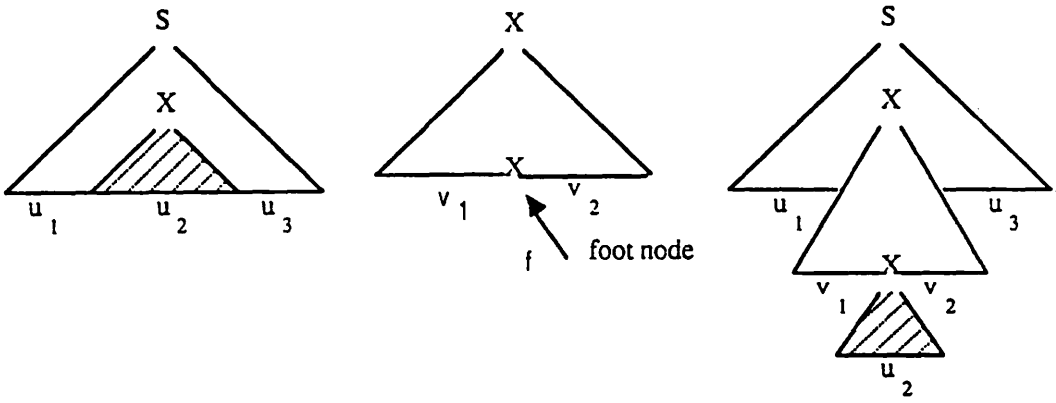

Figure 1.3: The operation of adjoining

β_1 in the elementary tree α_1. The node in α_1 where adjoining takes place is marked with an asterisk.

So far, the only restriction we have placed on the set of auxiliary trees that can be adjoined at a node is that the label of the node must be the same as the label of the root (and the foot) node of the auxiliary tree. Further restriction on this set of auxiliary trees, based on the so-called proper analysis context and domination contexts, was investigated by Joshi [Joshi 85]. But the full power of this definition was never used, and the context needed to determine the choice of trees for adjunction was localized within an elementary tree. In the present definition of TAG's, these local constraints are specified by enumerating with each node the subset of auxiliary trees which can be adjoined at that node. This enumeration of the set of auxiliary trees is called the *Selective Adjoining* (SA) constraints. If the set is an empty set then we say that the node has a *Null Adjoining* (NA) constraint. In general, it is understood that any of the trees in the selective adjoining

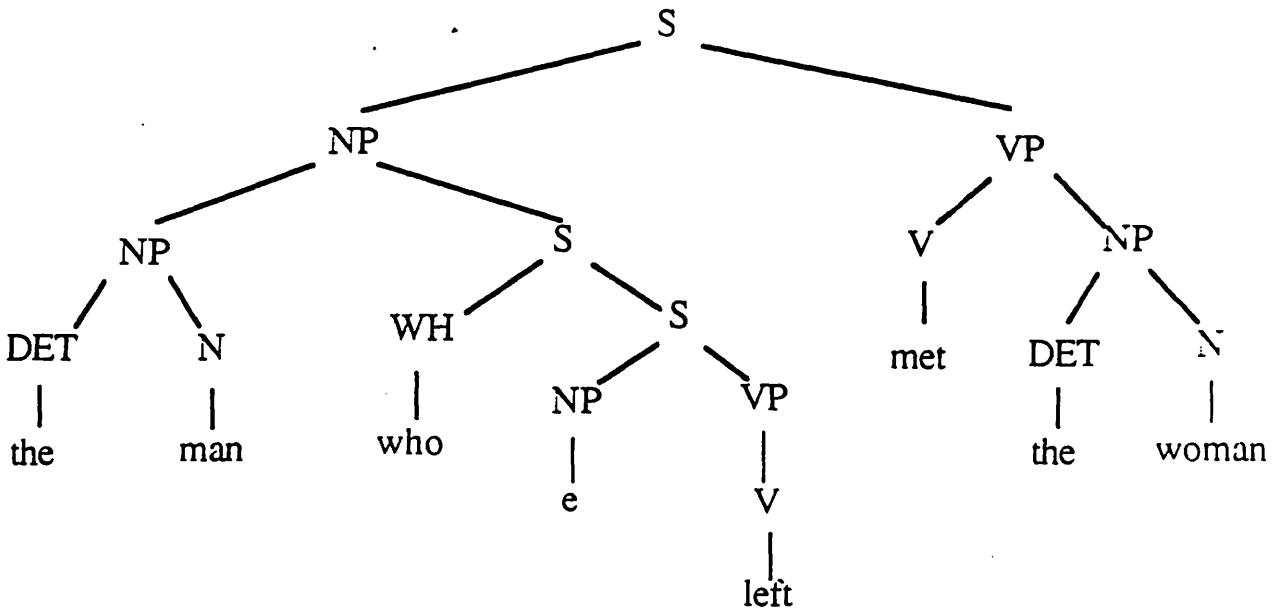

Figure 1.4: Adjoining β_1 in α_1

constraints of a node can be used for adjunction. It is possible to insist that adjunction is mandatory at a node. In such a case, we say that the node has an *Obligatory Adjoining* (OA) constraint. Consider the initial tree given in Figure 1.5. The sentential tree is not complete yet as it is an untensed sentence. However, on adjunction of β_2 at the root of α_2 (as shown in Figure 1.5), we obtain a complete sentence. Therefore, we say that the root of α_2 has an OA constraint.

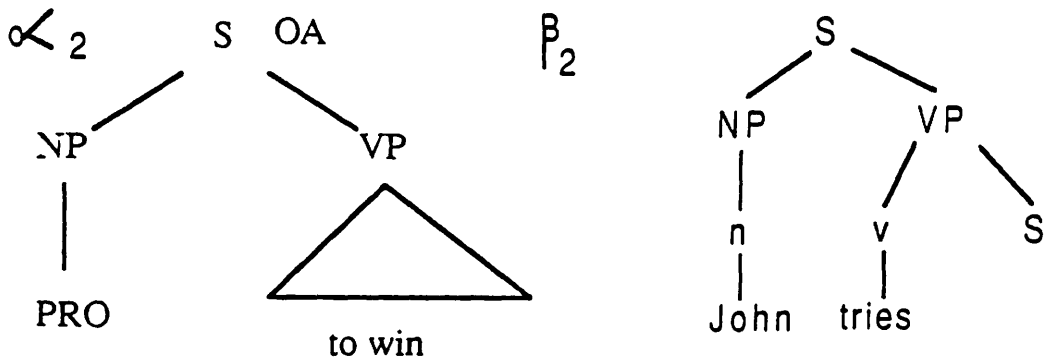

Figure 1.5: Use of obligatory adjoining constraint

In a TAG, the standard (capturing the intuitive notion of derivation) way of derivation

that has been used thus far proceeds as follows. We initially start with an initial tree and allow adjunction by an auxiliary tree. The sentential tree that is derived may receive further adjunctions by auxiliary tree. At the end of the derivation we expect the final tree to have no nodes with an OA constraint.

We have so far discussed the local constraints at the nodes of elementary trees. We use Figure 1.6 to illustrate how constraints are propogated to the nodes of derived trees. As shown in Figure 1.6, the adjunction of β is allowed only if β is a member of the set C. In the resulting tree the nodes corresponding to that contributed by γ (nodes in the unshaded part) have the same constraint as the corresponding nodes in γ. However, the nodes introduced due to β (nodes in the shaded part) are given the constraints of the corresponding nodes in β.

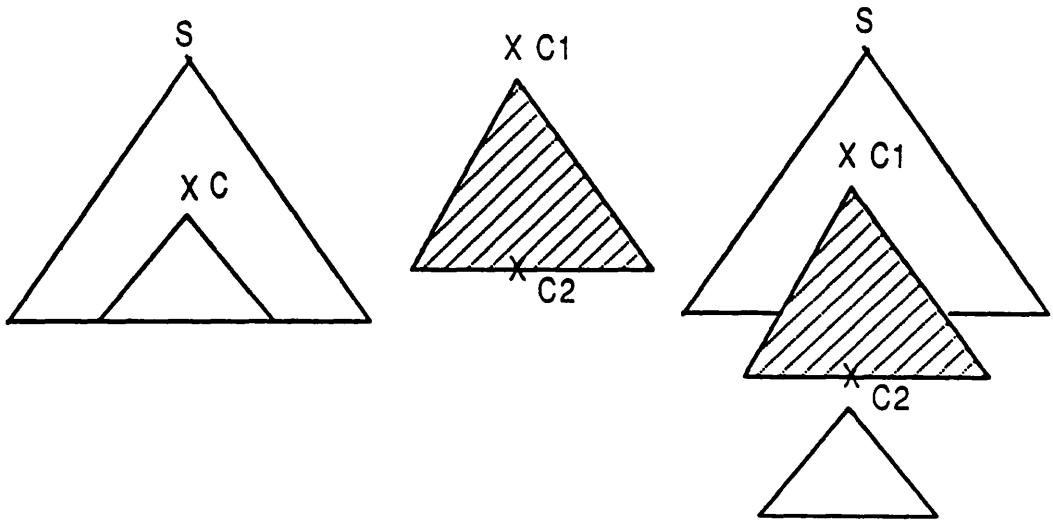

Figure 1.6: Propagation of Constraints

The tree set generated by a TAG is the set of sentential trees that do not have nodes with OA constraints. The string language corresponds to the terminal strings at the frontier of the sentential trees in the tree set generated by the TAG. The formal definition of TAG's is given in Section 1.4.

1.2.1 A Note on the Linguistic Theory underlying Tree Adjoining Grammars

We now describe some of the salient points of the theory that is used currently while designing TAG's for natural languages. The most important point of this theory is that it treats the principles of stating co-occurence facts stated on simple sentences as orthogonal to the those governing the generation of complex sentences. Thus, it is assumed that the elementary trees of a TAG are minimal linguistic structures. The dependencies are localized as far as possible to these elementary trees. Thus the elements that are dependent on each other due to some co-occurence relation are part of the same elementary tree. Thus, for example, the filler and the gap belong to the same tree. The elements that are subcategorized by a verb belong to the same elementary tree to which the verb belongs. A variety of constraints can be checked easily because the entire elementary tree, that is the domain of the constraints, is available as a single unit at each step of the derivation. Thus, the lexical insertion in the tree α_3 (given in Figure 1.7) violates the agreement constraints. The lexical insertion in the initial tree α_4, shown in Figure 1.7, is not an acceptable, since subcategorization constrints of the intransitive verb, *fell*, are violated. This principle is also used in the statemnt of the local constraints. Thus, the structure of α_2, given in Figure 1.5, is enough to specify that the SA constraint at the root allows only trees such as β_2 (which have its main verb subcategorizing for an infinitival sentence) to be adjoined.

Complete localization is not always possible. For example, the verb in the relative clause structure given by β_1 in Figure 1.2 agrees in number with the NP node it gets adjoined at (e.g., the subject NP of the initial tree α_1 as in Figure 1.3).

In a TAG, the dependencies are localized, and adjunction simply preserves these relations. Thus, another stipulation made in a TAG is that the grammatical relations (according to the tree derived) must not be disrupted due to an adjunction. For example

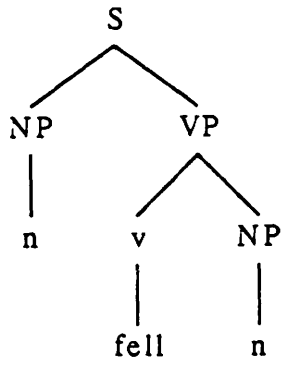

Figure 1.7: Unacceptable lexical insertion

in the derivation of the sentence

Mary thought John saw Bill hit Jill

the derivation given in Figure 1.8 is not acceptable, since in the intermediate tree obtained

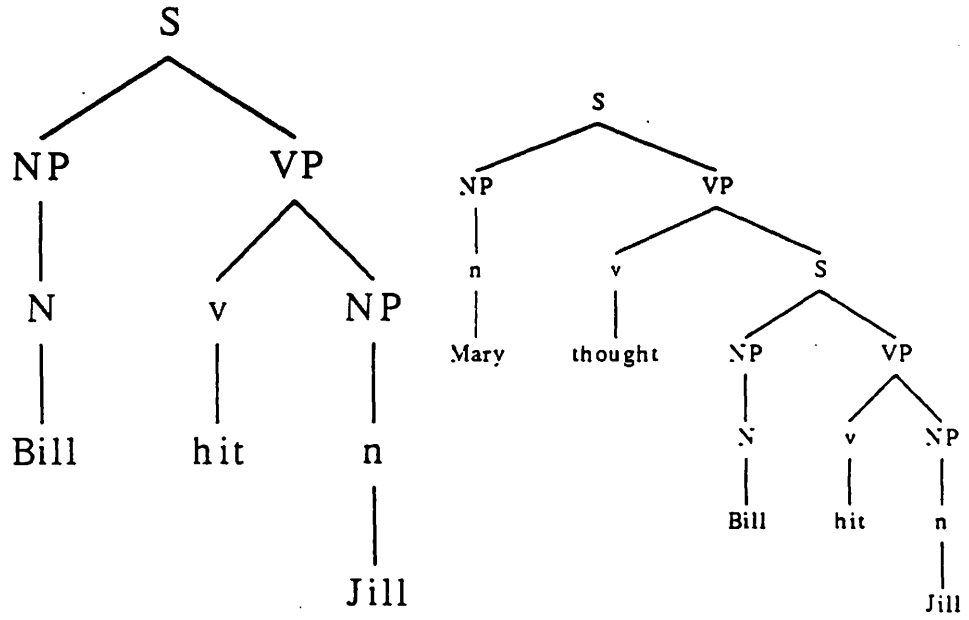

Figure 1.8: An unacceptable sequence of adjunctions

we expect the proposition that Mary thought Bill hit Jill to hold. In a TAG, we will insist that the foot node of auxiliary trees such as β_4 to have an NA constraint. Thus, from the intermediate tree corresponding to

Mary thought Bill hit Jill

the sentence given above can not be derived, and the only sequence allowed is the one which derives the intermediate sentence

John saw Bill hit Jill.

1.3 A Brief Description of the Results in the Thesis

In Section 1.4, we give a formal definition of TAG's. We introduce the notion of *derivation trees* which may be used to capture the derivation history in a TAG.

Pollard [Pollard 84] introduced a grammatical formalisms, called Head Grammars (HG), which is an extension of Context-Free Grammars. This formalism has an operation of *wrapping* strings in addition to the concatenation operation on strings. These operations are partial functions, not defined when one of the arguments is the empty string. Based on a suggestion made in [Roach 84] (which studies the formal properties of Head Grammars), we alter the definition to allow the operations to be defined on all strings. This formalism, which we call Modified Head Grammars, is shown to be equivalent to TAG's in Chapter 2.

In Section 3.3, we consider a formalism which is a linearized version of Indexed Grammars. This formalism was studied by Gazdar [Gazdar 85a] for its linguistic applicability. We show that this formalism is equivalent to TAG's in Section 3.3.

Joshi [Joshi 85] has a characterized a class of formalisms called *mildly context-sensitive formalisms* by a giving a set of properties that should hold of the formalisms and the languages they generate. Although this definition is not formalized, it was an attempt to characterize the properties of formalisms that interesting in the context of constrained formalisms for natural languages. Tree Adjoining Grammars, Head Grammars and Linear Indexed Grammars, the three systems whose equivalences we have considered, satisfy these set of properties. Although the three formalisms are based on different linguistic motivations and ideas, and have varying notations, the results showing the equivalences may provide insight into the nature of the structures in natural languages besides the insights provided by the individual systems. It is in this light that we would like to view our results on equivalences.

In Chapter 3, we give the definition of an *embedded pushdown automaton* and show

that it is equivalent to TAG's. This automaton characterization of TAG's gives us insight into the power of TAG's, and why it generalizes CFG's and yet is constrained when compared to Indexed Grammars.

In Chapter 4, we prove a few results about TAL's. First, in Section 4.1, we study the closure results of TAL's and show that they form a substitution closed Full Abstract Family of Languages (Full AFL). In Section 4.2, we give a pumping lemma for TAL's adapting the $uvwxy$ theorem for CFL's. The constrained nature of TAL's is further observed when we show that every TAL is a semilinear language. The question of whether the property of semilinearity (and a closely related property of linear growth) is relevant in the study of natural languages has been addressed in [Joshi 85,Berwick 84]. It is postulated that the growth of structures in the derivation of natural language sentences is linear. Semilinearity and the linear growth properties are approximations of this property.

In Section 4.4, we give a parsing algorithm for TAL's which works in $O(n^5)$ time where n is the length of the input. The space complexity is $O(n^4)$.

In Chapter 5, we develop a formalism called Feature Structure based Tree Adjoining Grammars (FTAG). Several grammatical formalisms incorporate the notion of feature structures and the unification operation on them, to specify the co-occurence constraints of features of dependent items. FTAG embeds TAG's in such a framework. In Section 5.2.5, we give a denotational semantics of the formalism. We show that FTAG is useful in the description of natural languages, avoiding the specification of local constraints as they are present in TAG's. We are also able to avoid duplication of trees needed in the TAG formalism to handle some constructions. We show, in Section 5.2.7, that FTAG in its full generality is a very powerful tool, and propose a restricted use of feature structures (in Section 5.3) in order to make the parsing problem more tractable.

Finally, in Chapter 6, we make summarize our study of Tree Adjoining Grammars, and discuss some problems which arise out of our work.

1.4 Definitions of Tree Adjoining Grammar

In this section, we will define Tree Adjoining Grammars (TAG) and give a normal form for TAG's. But first, we define a notation for trees. This notation is due to Gorn [Gorn 62]. Let $\mathcal{N}^*$ be the free monoid generated by $\mathcal{N}$ (the set of natural numbers), whose binary operation is given by $\cdot$ and identity is ϵ. For $p, q \in \mathcal{N}^*$, $p \leq q$ iff there is a r $\in \mathcal{N}^*$ such that $q = p \cdot r$. $p < q$ iff $p \leq q$ and $p \neq q$.

Definition 1.1 Definition of a Tree.

The addressing scheme for nodes of a tree is as follows. The address of the root of a tree is given by ϵ. The addresses $i \cdot 1, \cdots, i \cdot j$ represent the j children (in the left to right order) of the node addressed i.

For every tree, γ, the tree domain of γ, D_γ, is is a finite subset of $\mathcal{N}^*$ such that the following holds.

- if $q \in D_\gamma, p < q$ then $p \in D_\gamma$.
- if $p \cdot j \in D_\gamma, j \in \mathcal{N}$, then $p \cdot 1, \cdots, p \cdot (j-1) \in D_\gamma$

If V is finite alphabet used to label the nodes of trees, then γ is a tree over V if it is a function from D_γ into V.

We shall now define subtrees and supertrees.

Definition 1.2 Subtree and Supertree.

Let γ be a tree and $p \in D_\gamma$ then,

$$\gamma/p = \{< q, A >|< p \cdot q, A >\in \gamma, q \in N^*\},$$
$$\gamma\backslash p = \{< q, A >|< q, A >\in \gamma, p \not\leq q\}$$

are subtree and supertree at p respectively.

If γ is a tree, then we use the notation $< \gamma, i >$ to represent the node in γ with address i.

Definition 1.3 Preliminary Definition of TAG's

A Tree Adjoining Grammar G is a quintuple (N, Σ, S, I, A) where

N is the finite set of nonterminal symbols

Σ is the finite set of terminal symbols

I is the finite set of *initial* trees

A is the finite set of *auxiliary* trees

S is a distinguished nonterminal symbol called the start symbol.

An *initial* tree is a tree with the label of the root being S, and the yield belonging to Σ^*.

An *auxiliary* tree is a finite tree whose root is labelled with a nonterminal symbol. A node in the frontier called the foot node is labelled by the same nonterminal symbol. The yield belongs $\Sigma^*N\Sigma^*$.

A tree is called an *elementary* tree if it is an initial tree or an auxiliary tree.

Definition 1.4 Preliminary Definition of Adjoining

Let i be the address of a node in some tree γ. Let this node be labelled by a nonterminal symbol X. Then we can adjoin an auxiliary tree β whose root node (and foot node) is

labelled with X. Let j be the address of the foot node of β. Adjoining β at the node given by address i gives rise to the tree represented as $\gamma[i, \beta]$.

$$\gamma[i, \beta] \triangleq \gamma \backslash i \cup i \cdot \beta \cup (i \cdot j) \cdot (\gamma / i)$$

where we define $i \cdot \beta$ as $\{< i \cdot k, A > | < k, A > \in \beta\}$. Similarly, $(i \cdot j) \cdot (\gamma / i) = \{< i \cdot j \cdot k, A > | < k, A > \in \gamma / i\}$. The effect of adjoining an auxiliary tree, β, is shown in the figure 1.3. Note that the definition of adjoining does not specify that the labels of the root and foot nodes of the tree used for adjunction should match with the label of the node where adjunction takes place. This is a requirement placed by TAG's and not by the definition of adjoining.

1.4.1 Constraints on Adjoining

In the previous definition, we said that any auxiliary tree can be adjoined at a node provided the labels at the node and the root (and hence the foot node) are the same. However, we can further restrict the choice of auxiliary trees that can be used for adjunction at a node. With every node we specify a set of auxiliary trees (whose root symbol is the same as the label of this node). Only one of these trees may be used for adjoining at this node. We call the set of auxiliary trees associated with the node as the *Selective Adjoining constraints* (SA constraints) of the node. If the set is an empty set, then it was termed the *Null Adjoining constraints* (NA constraints). We specify the SA constraints by a function, denoted by $\mathcal{C}$, specifying the mapping from nodes to these sets of trees. Further, we also specify whether adjunction is obligatory at a particular node. This is specified by the function $\mathcal{O}$ mapping the nodes to the boolean set. We say that a node has *Obligatory Adjoining constraints* (OA constraints) when $\mathcal{O}$ specified that adjunction is compulsory at a node.

On adjunction, the nodes in the resulting tree have to be assigned constraints. This is

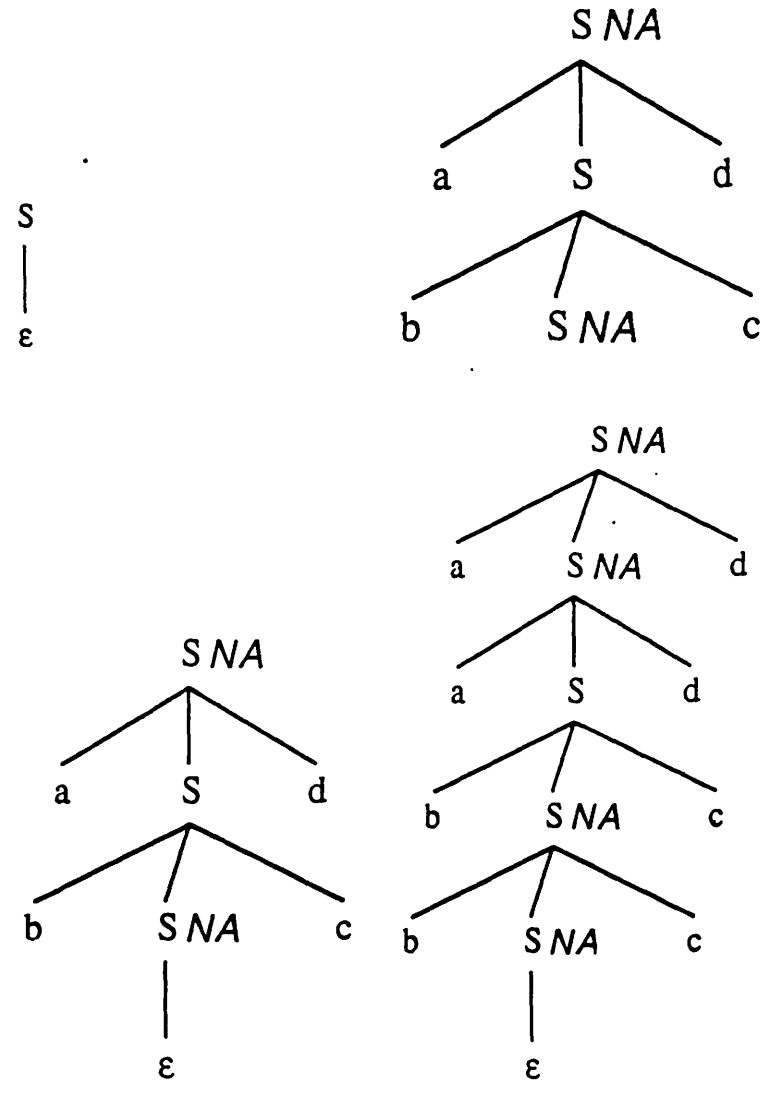

Figure 1.9: A Sample Grammar

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

We generalize the notion of sentential trees, and describe *derived trees*. Derived trees are those which can be derived starting from any elementary tree, not necessarily from an initial tree.

- An elementary tree, γ, itself is a tree derived from γ, written as $\gamma \in D(\gamma)$.

- Adjoining into a derived tree in $\mathcal{D}(\gamma)$ yields a derived tree in $\mathcal{D}(\gamma)$.

Notice that a tree, γ, derived from an auxiliary tree, β, will be similar to β in that it will have a root and foot node with the same label as the root and foot node of β.

The definition of adjoining can itself be generalized to allow adjoining by derived auxiliary trees. If η is a node in some derived tree γ (represented as $< \gamma, i >$), and $\mathcal{C}(\eta)$ includes β then any derived auxiliary tree $\gamma' \in \mathcal{D}(\beta)$ can be used for adjoining at η. The rest of the definition remains the same. Note a tree is a sentential tree if it belongs to $\mathcal{D}(\alpha)$ for some initial tree α.

Generalizing further, we can talk of trees that are derived from subtrees of elementary trees. Thus if, γ is an elementary tree with a node addressed i, then the definition of trees derived from the subtree rooted at i is given by $\mathcal{D}(\langle \gamma, i \rangle)$.

- The subtree, γ/i, is a member of $\mathcal{D}(\langle \gamma, i \rangle)$.
- Adjoining into a derived tree in $\mathcal{D}(\langle \gamma, i \rangle)$ yields a derived tree in $\mathcal{D}(\langle \gamma, i \rangle)$.

We now present a scheme of presenting a derivation history in which derivation proceeds *inside out*. This gives a canonical way of defining a derivation sequence which we will use extensively in showing some properties of TAL's. In this scheme, adjoining is allowed only in the nodes of elementary trees. But we allow the adjunction of derived auxiliary trees in an elementary tree. The derivations may be thought as occuring in the following sequence. The trees are built inside out. First, all the adjunctions take place in auxiliary trees, deriving derived auxiliary trees. These derived auxiliary trees are then used in adjunction into elementary trees to give new derived elementary trees. This process continues until we adjoin into initial tree. Thus, the derivation in Figure 1.9. may be represented by adjoining β into itself and using the derived auxiliary tree into the initial tree as shown in the figure 1.10.

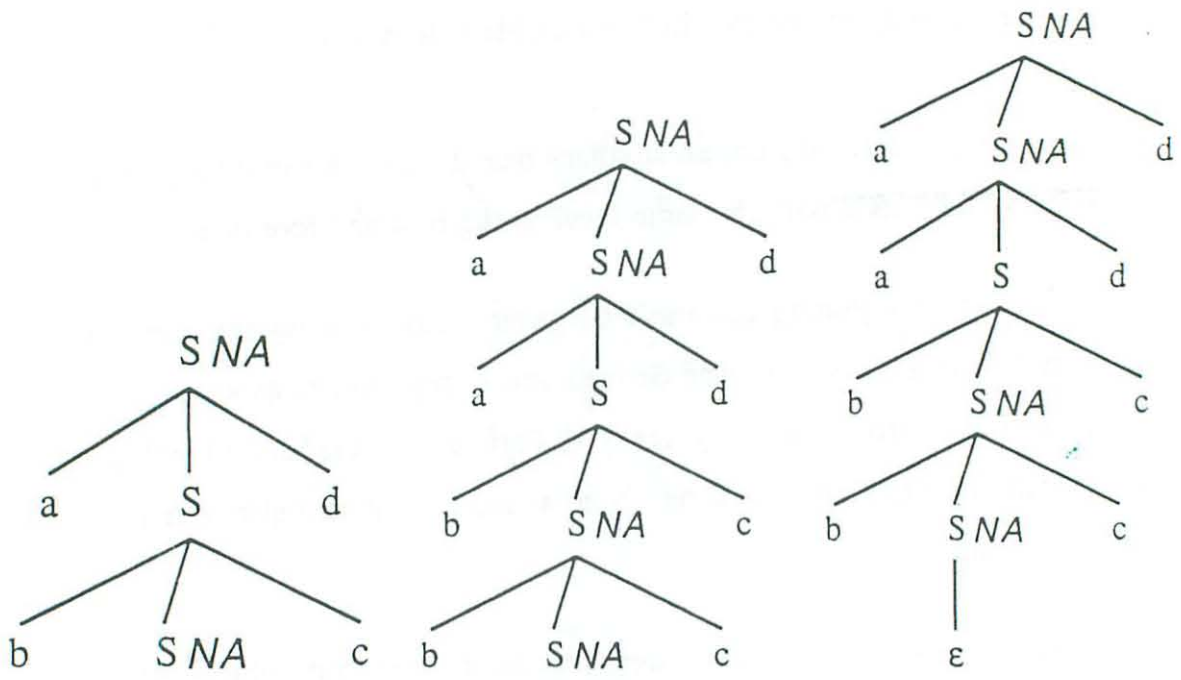

Figure 1.10: An insideout derivation

We now give a method of presenting a derivation history of trees generated by TAG's. These derivation histories will be given by trees which tell us which trees were adjoined and their sequence, and will be called *derivation trees*. In CFG's, the derived tree itself gives us the derivation history. Thus, the derivation tree is the same as derived trees. The derivation trees of TAG's will be similar to those of CFG's. Each level of the derivation trees of CFG's describe the combination of symbols manipulated according to the rules of the grammar. To give an analogous description for TAG's, we have to give rules of the grammar describing the combination of objects in a TAG. The objects that are composed by a TAG are the elementary trees as well as derived trees. But note that for each elementary tree, γ, $\mathcal{D}(\gamma)$ gives us the set of trees generated from γ. Thus, analogous to the concept of nonterminals in a CFG, if γ is an elementary tree, the symbol $\overline{\gamma}$ is used to describe the set of trees derived from γ. Thus, if a derived auxiliary tree, derived from the auxiliary tree β, is adjoined at a node addressed i of an elementary tree

γ[1], then this derivation is captured by a rule of the form

$$\overline{\gamma} \rightarrow A(\gamma, \langle \overline{\beta}, i \rangle \cdot)$$

In this scheme, we wish to give a set of rules which can be used to capture all the possible derivations. For this to be possible, we can only use the addresses of nodes in elementary trees in the rules, and not those of derived trees (as they are not known before their derivation). Thus, if there is a sequence of adjunctions which take place at distinct nodes of some elementary tree, we must consider all the adjunctions to have been done in parallel. Then, all the addresses used in these rules will be those of elementary trees. The derivation sequence given by the adjunction of $\beta_1, \ldots, \beta_k$ (or trees derived from them) at *distinct* nodes addressed $i_1, \ldots, i_k$ of an elementary tree γ is given by the rule

$$\overline{\gamma} \rightarrow A(\gamma, \langle \overline{\beta_1}, i_1 \rangle, \ldots, \langle \overline{\beta_k}, i_k \rangle)$$

Since, mentioning γ and A is redundant, we sometimes represent this rule in one of the following two ways.

$$\overline{\gamma} \rightarrow \langle \overline{\beta_1}, i_1 \rangle, \ldots, \langle \overline{\beta_k}, i_k \rangle \text{ or}$$
$$\overline{\gamma} \rightarrow \overline{\beta_1} @ i_1, \ldots, \overline{\beta_k} @ i_k$$

If in a derivation sequence, no tree is adjoined in an elementary tree γ, we represent this derivation as $\overline{\gamma} \rightarrow \epsilon$ or $\overline{\gamma} \rightarrow o$. Note that in any derivation involving a tree γ, every node with OA constraints has to receive an adjunction. Thus, we stipulate that every rule for the derivations from γ must include the addresses of all the nodes in γ with OA constraints.

Considering an elementary tree and the constraints at nodes of this tree, we can write a set of rules to capture all the possible derivation sequences of derived trees starting from this elementary tree. For example, let γ is an elementary tree, and $\{i_1, \ldots, i_n\}$ be

[1] if β can be adjoined at a node, then every tree derived from β can be adjoined at the same node

the addresses of a subset of nodes of γ (which includes every node of γ with an OA constraint). Let β_k be adjoinable at the node in γ addressed i_k $(1 \leq k \leq n)$. Then we will have a rule

$$\overline{\gamma} \rightarrow \overline{\beta_1}@i_1, \ldots, \overline{\beta_n}@i_n$$

The collection of the set of rules for all the elementary trees of a TAG, G, is called the *derivation rules of G*.

Given the derivation rules of a TAG, we can represent derivations by *derivation trees*. The derivation trees of a TAG are like the derivation trees of CFG except the edges of the tree are labelled by the addresses where adjunction takes place and the frontier is a series of nodes labelled by ϵ (or ϕ).

Derivation trees are now defined formally.

- if γ is an elementary tree with no OA constraints, then the following tree is a derivation tree

Figure 1.11: Definition of derivation trees - base case

- If Υ_j is a derivation tree with the root labelled $\overline{\beta_j}$ for $1 \leq j \leq k$, then given the derivation rule

$$\overline{\gamma} \rightarrow \overline{\beta_1}@i_1, \ldots, \overline{\beta_k}@i_k$$

the following is also a deri ation tree.

If G is a TAG, then the set of derivation trees is denoted by $T_D(G)$.

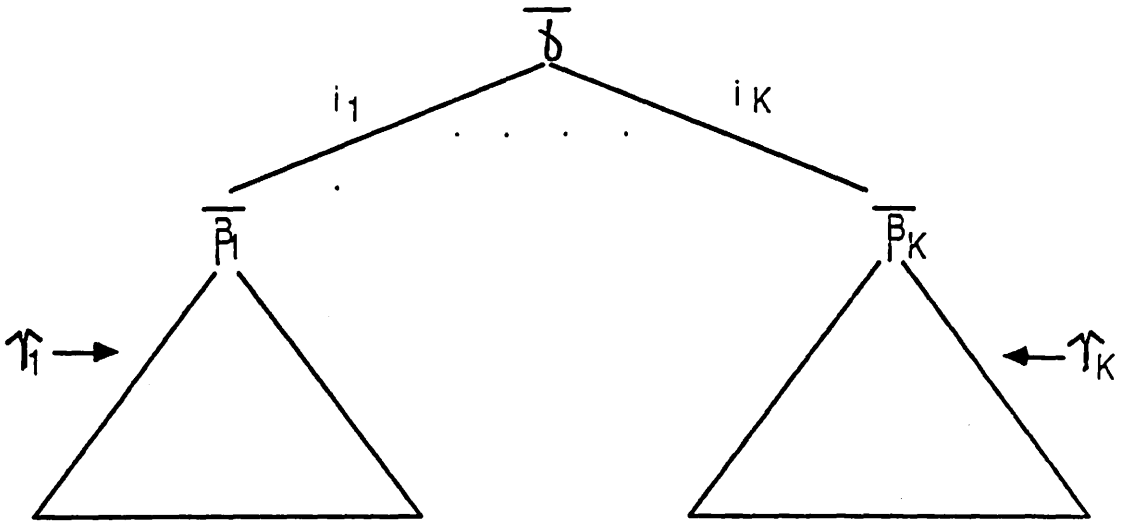

Figure 1.12: Definition of derivation trees - inductive case

As an example, we can consider the derivation of the tree in Figure 1.10 and give its derivation tree. Note that every derivation tree Υ[2] corresponds to a derived tree - which

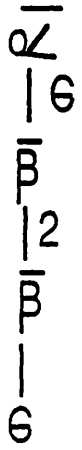

Figure 1.13: An example derivation tree

has no OA constraints. We can obtain γ by traversing it in bottom-up or top-down fashion, and performing the required adjunctions at the appropriate nodes. In this case, we say that γ *is described by* Υ.

If i is the address of a node in an elementary tree γ, then we can give the derivation

[2] We use the symbols $\Upsilon, \Upsilon_1, \ldots$ for derivation trees

trees of trees in $\mathcal{D}(\langle \gamma, i \rangle)$ by considering derivation trees whose root symbol is $\overline{\gamma}$ and ignoring the adjunctions at nodes other than i and its descendants.

We have seen that every derivation tree describes a derived tree with no nodes having OA constraints. We now show that for every tree γ which has no OA constraints, there is a derivation tree Υ which describes it. Thus, we will show that for every tree in $T(G)$ derived in the standard way (i.e., starting from an initial tree), there is a derivation tree that describes it. Note in that the standard derivations, we could have built trees with OA constraints. Therefore, to describe the derivation of these partially built trees, we will define *incomplete derivation trees*. Suppose we are considering the derivation tree in a top-down fashion, and that we have a node labelled $\overline{\gamma}$. If a node addressed i in γ has an OA constraint, then every rule which expands $\overline{\gamma}$ is of the form

$$\overline{\gamma} \rightarrow \upsilon_1, \beta @ i, \upsilon_2$$

for some υ_1, υ_2 and β adjoinable at the node addressed i in γ. Consider the derivation tree which specifies that β has been adjoined at this node. If we want to describe the tree where adjunction at this node, with the OA constraint, has not taken place. This can be described by the derivation tree the subtree below the edge labelled i (corresponding to the adjunction of the tree derived from β) is removed. Such a tree is incomplete since it does not have the information about the subtree below the edge labelled i. This tree can be completed if and only if a derivation tree is inserted below the dangling edge. This will not be the case if the node addressed i does not have an OA constraint. A tree with m incomplete edges is called an $m - incomplete$ derivation tree. Note that a 0-incomplete derivation tree is a derivation tree.

Definition 1.8 m-Incomplete Derivation Trees

- Any derivation tree is an 0-incomplete derivation tree.

- Let γ be an elementary tree. Let $\{i_1, \ldots, i_k\}$ $(k \geq 0)$ be the addresses of a subset of nodes in γ with OA constraints. Let $\{n_1, \ldots, n_q\}$ be the addresses of the other nodes in γ with OA constraint. Let $\{j_1, \ldots, j_l\}$ $(l \geq 0)$ be the addresses of a subset of nodes in γ not having OA constraints. Let the auxiliary tree β_{j_p} be adjoinable at the node addressed j_p in γ and Υ_{j_p} be a m_{j_p}-incomplete derivation tree with the root symbol $\overline{\beta_{j_p}}$ for $1 \leq p \leq l$. Similarly, let the auxiliary tree β_{n_p} be adjoinable at the node addressed n_p in γ and Υ_{n_p} be a m_{n_p}-incomplete derivation tree with the root symbol $\overline{\beta_{n_p}}$ for $1 \leq p \leq q$. Then the following tree is an $m = (k + m_{j_1} + \cdot + m_{j_l} + m_{n_1} + \cdot + m_{n_q})$-incomplete derivation tree.

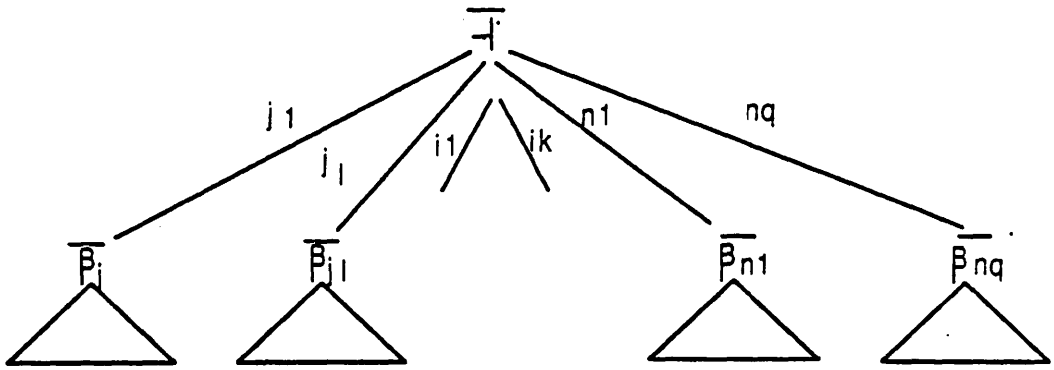

Figure 1.14: An m-incomplete derivation tree

Theorem 1.1 For every tree γ derived (using the most general definition of derivation in TAG's) with k $(k \geq 0)$ adjunctions such that there are m nodes in γ with OA constraints, there is a m-incomplete derivation tree Υ which describes γ.

We prove this result by inducting on the number of adjunctions performed in deriving γ.

Base Case: $k = 0$

Then γ is an elementary tree which has m nodes with OA constraints. The required derivation tree is a tree has one node labelled $\overline{\gamma}$ with m edges dangling labelled by addresses of the nodes with OA constraints.

Inductive Step:

Assume the statement of the theorem holds for all $k' < k$. Now let γ be derived using

k adjunctions and that there are m nodes in γ with OA constaints. We can assume that some derived auxiliary tree γ_1 was adjoined at a node η in a derived tree γ_2 to give γ. Let us assume the number of adjunctions in γ_1, γ_2 are k_1, k_2 respectively and that γ_1, γ_2 have m_1, m_2 (respectively) nodes with OA constraints. Thus, $k = k_1 + k_2 + 1$. $m = m_1 + m_2$ or $m = m_1 + m_2 - 1$ depending on whether the node η in γ had an OA constraint. The inductive hypothesis holds for the derivations of γ_1 and γ_2. Let the m_1-incomplete derivation tree describing γ_1 (respectively m_2-incomplete derivation tree describing γ_2) be called Υ_1 (respectively Υ_2). Our goal now is to find the m-incomplete derivation tree Υ which describes γ. The first step is to find out which elementary tree, γ_3, contributed the node η in γ_2 and where this instance of γ_3 appears in Υ_2 so that we can insert Υ_1 below it.

The identification of γ_3 and its location in Υ_2 is done using the following *procedure*. Using any normal order (say prefix) traversal procedure, traverse Υ_2 building up γ' in the following way. If the traversal leads us to a node in the derivation tree (addressed p) such that we have to consider adjoining β at some node addressed i in an elementary tree γ'', adjoin a tree which has the same structure as β except that the nodes are now labeled (β, j, X, p) where the node addressed j in the auxiliary tree β is labelled X. Note that γ' will have the same structure as γ_1 except for the fact that the labels of nodes in γ' are given by the $4-tuple$. We can have a 1-1 mapping from the nodes of γ' to the nodes of γ_1 by mapping the 4-tuple to the label (the 3^{rd} element of the tuple) of nodes in γ_1. We can then find the node in γ'' which corresponds to η. Let this node be labeled by the 4-tuple (γ'', j, X, p). Then Υ is obtained by inserting the m_1-incomplete derivation tree Υ_1 in the m_2-incomplete derivation tree Υ_2 below the node which is $(p)^{th}$ node visited in the prefix traversal of Υ_2 (then p is its address). There are two cases to consider
1). There is an edge dangling below this node which is labeled j. This case then corresponds to $m = m_1 + m_2 - 1$. We insert Υ_2 below this node and complete this edge.
2) This case corresponds to η not having an OA constraint. If the derivation rule used at

the p^{th} node was

$$\gamma'' \rightarrow \beta_1 @ i_1, \ldots, \beta_n @ i_n \qquad (I)$$

then $j \notin \{i_1, \ldots, i_n\}$, otherwise β could not have been adjoined at η. Note then that there is also a derivation rule

$$\gamma'' \rightarrow \beta_1 @ i_1, \ldots, \beta_n @ i_n, \beta @ j \qquad (II)$$

Then, Υ is the same as Υ_1 except that the derivation rule (II) is used instead of (I) and Υ_2 is inserted below the link labeled by the address j. Υ then describes γ as required.

1.4.2 A Normal form for Tree Adjoining Grammars

We now give a simple normal form for TAG grammars which simplify the proofs while proving some properties of TAL's. In this normal form we have only a single initial tree.

Theorem 1.2

For every TAG, $G = (N, \Sigma, S, I, A)$ there is a TAG, $G_1 = (N, \Sigma, S, \{\alpha\}, A_1)$ such that $L(G) = L(G_1)$.

For each α in I, we have an auxiliary tree, β_α, as shown in Figure 1.15. We let $A_1 = A \cup \{\beta_\alpha \mid \alpha \in I\}$. α_1, the only initial tree in G_1 is given in Figure 1.15

It is straightforward to show that $L(G) = L(G_1)$. For every $\beta \in A$, if Υ is a derivation tree such that the root is labeled by $\overline{\beta}$ then Υ also belongs to $T_D(G_1)$. Now considering a derivation tree, Υ_1, in $T_D(G)$ given in Figure 1.16, there is a derivation tree, Υ_2, in $T_D(G_1)$ (as shown in Figure 1.16), such that both describe trees with the same frontier. Thus, $L(G) = L(G_1)$.

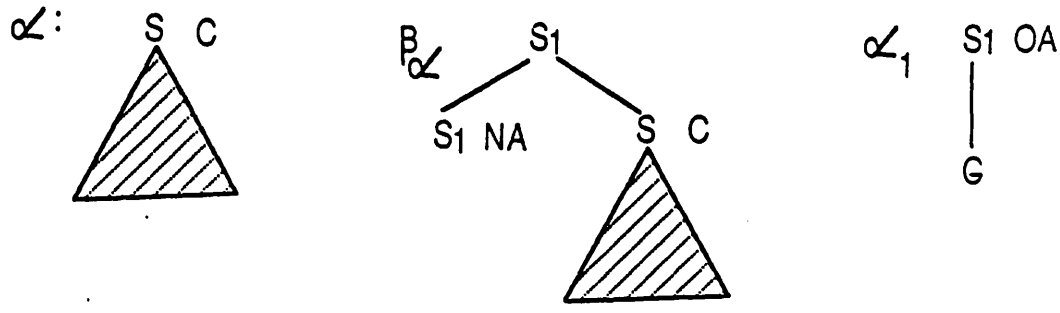

Figure 1.15: Single initial tree normal form

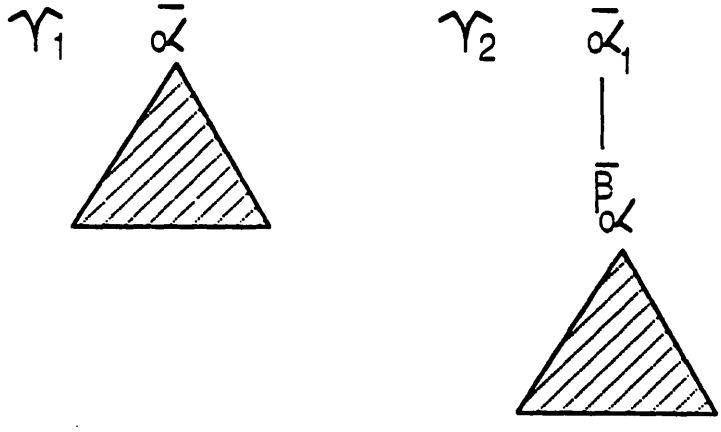

Figure 1.16: Equivalence of G and G_1

Chapter 2

Tree Adjoining Grammars and Head Grammars

In this chapter we discuss the relationship between TAG's and Head Grammars (HG's). HG's were introduced by Pollard [Pollard 84] to capture certain structural properties of natural languages; HG's have a completely different motivation from that of TAG's. These formalisms are notationally quite different. HG's maintain the essential character of context-free string rewriting rules, except for the fact that in addition to concatenation of strings, string **wrapping** operations are permitted. Formal properties of HG's were investigated by Roach [Roach 84]. It was observed that the two systems seemed to possess similar generative power and have the same closure properties [Roach 84,VijayShanker 85] as well as similar parsing algorithms [Pollard 84,VijayShanker 85], a significant amount of indirect evidence existed to suggest that they were formally equivalent. In this chapter, we will discuss the relationship between HG's and TAG's. The complete details can be found in [Joshi 86,VijayShanker 86b]. The linguistic implications of this study are in [Weir 86].

Vijay-Shanker and Joshi [VijayShanker 85] provided a brief description of the intu-

ition behind the inclusion of Tree Adjoining Langages (TAL) in the class of languages generated by a variant of HG's called Modified Head Grammars (MHG). We give a proof of this result as well as a proof for the inclusion of Modified Head Languages (MHL) in TAL, hence showing that MHG's and TAG's are equivalent. This result is presented in section 2.2. In Section 2.2.4, we discuss the relationship between HL's and MHL's, and hence the relationship between TAL's and HL's.

Before presenting the results of this section, we first define the grammatical formalisms. At the end of section ??, we present a brief explanation of the relationship between the operations of string wrapping and tree adjunction.

2.1 Head Grammars and Modified Head Grammars

In this section we define Head Grammars and Modified Head Grammars and describe how they generate string languages.

2.1.1 Definition of Head Grammars

We first introduce the notion of a **headed string**. A headed string is a string of symbols containing one distinguished symbol referred to as the *head* of the string. Formally, this can be represented as a pair consisting of a string w and an integer that indicates the position of the head in the string. When we wish to explicitly mention the head, we use the representation $w_1\overline{a}w_2$ where $w_1aw_2 = w$; alternatively, we can denote the he ıded string by $\overline{w}$. This allows us to denote the head·d empty string as $\overline{\lambda}$. Huh?

In a Head Crammar (HG), productions are like those in a CFG except that ins ı ad of just concatenating the strings derived from the constituents, we may use another st ng operation called the wrapping operation. Further the rules will also have to state wh ch

constituent's head is the head of the resulting phrase.

Productions are of the form: $A \rightarrow f(\alpha_1, \ldots, \alpha_n)$ or $A \rightarrow \alpha_1$ where $A \in V_N$, α_i either belongs to V_N or is a headed string and

$$f \in \bigcup_{i \geq 1} \{ LCi, LLi, LRi, RCi, RLi, RRi \}$$

Roach [r84] has shown that there is a normal form for Head Grammars which only uses the operations given below. Definitions of the other operations can be found in [Pollard 84].

The language generated by a HG G is defined as follows:

$$L(G) = \{ w \mid S \stackrel{*}{\underset{G}{\Longrightarrow}} \overline{w} \}$$

We now define the 6 operations that were mentioned above.

$$\begin{aligned}
LC1(u_1\overline{a_1}u_2, v_1\overline{b_1}v_2) &= u_1\overline{a_1}u_2, v_1b_1v_2, \\
LC2(u_1\overline{a_1}u_2, v_1\overline{b_1}v_2) &= u_1a_1u_2, v_1\overline{b_1}v_2, \\
LL1(u_1\overline{a_1}u_2, v_1\overline{b_1}v_2) &= u_1\overline{a_1}v_1b_1v_2u_2, \\
LL2(u_1\overline{a_1}u_2, v_1\overline{b_1}v_2) &= u_1a_1v_1\overline{b_1}v_2u_2, \\
LR1(u_1\overline{a_1}u_2, v_1\overline{b_1}v_2) &= u_1v_1b_1v_2\overline{a_1}u_2, \\
LR2(u_1\overline{a_1}u_2, v_1\overline{b_1}v_2) &= u_1v_1\overline{b_1}v_2a_1u_2
\end{aligned}$$

Both Pollard [Pollard 84] and Roach [Roach 84] have defined these operations as partial functions. Pollard's definition of headed strings includes the headed empty string ($\overline{\lambda}$). However, mathematically, these do not have the same status as other headed strings; for example, $LC1(\overline{\lambda}, \overline{w})$ is undefined. In general, the term $fi(\overline{w_1}, \ldots, \overline{w_i}, \ldots, \overline{w_n})$ is undefined when $\overline{w_i} = \overline{\lambda}$. This nonuniformity has led to difficulties in proving certain formal results about Head Grammars (discussed in Roach [Roach 84]), and has caused problems in showing the equivalence of MHG's and HG's (see [Joshi 86,VijayShanker 86b]). The relationship between MHG's and HG's is discussed later.

We now give a sample Head Grammar, G_1.

$$S \rightarrow LL2(S_1, \overline{f}) \qquad S \rightarrow LL2(S_2, \overline{f})$$
$$S_1 \rightarrow LC2(\overline{a}, S_3, d) \quad S_2 \rightarrow \overline{g}h$$
$$S_3 \rightarrow LL2(S_1, \overline{b}c) \quad S_3 \rightarrow LL2(S_2, \overline{b}c)$$

The language generated by this grammar is

$$L_1 = \{ a^n g b^n f c^n h d^n \mid n \geq 0 \}$$

2.1.2 Modified Head Grammars

We define a formalism which we shall call Modified Head Grammars (MHG's). This formalism closely resembles HG's. Instead of headed strings, MHG's have split strings. A split string has a distinguished position between two strings in V_T^*, about which it may be split. We will denote a split string as $w_1 \uparrow w_2$ where $w_1 w_2 \in V_T^*$. Notice that we can represent the split empty string as $\lambda \uparrow \lambda$, though this will be denoted by λ. In MHG's, there are three operations on split strings – W, $C1$ and $C2$, defined as follows:

$$W(w_1 \uparrow w_2, u_1 \uparrow u_2) = w_1 u_1 \uparrow u_2 w_2$$
$$C1(w_1 \uparrow w_2, u_1 \uparrow u_2) = w_1 \uparrow w_2 u_1 u_2$$
$$C2(w_1 \uparrow w_2, u_1 \uparrow u_2) = w_1 w_2 u_1 \uparrow u_2$$

The operations $C1$ and $C2$ correspond to the operations $LC1$ and $LC2$ in HG's. The operation W has been defined such that the split point of its second argument becomes the split point of the string resulting from application of the operation.

Since the split point is not a symbol but a position between strings, separate operations corresponding to $LL2$ and $LR2$ are not needed. In addition, unlike HG's, which distinguish the two wrapping operations $LL1$ and $LL2$, W suffices as a substitute for both of these operations. When we want the split point of the *first* argument to become

the split point of the derived string, we must also specify whether the split point should be to the right or left of the second argument. In either case, we can simulate such operations using W, $C1$, and $C2$. For example, suppose $Y \stackrel{*}{\underset{G}{\Rightarrow}} w_1 \uparrow w_2$ and $Z \stackrel{*}{\underset{G}{\Rightarrow}} u_1 \uparrow u_2$ and we want X to derive $w_1 \uparrow u_1 u_2 w_2$. This can be achieved using the following two productions: $Z^{fr} \rightarrow C1(\lambda, Z)$ and $X \rightarrow W(Y, Z^{fr})$.

Productions of a MHG are of the form: $A \rightarrow f(\alpha_1, \alpha_2)$ or $A \rightarrow \alpha_1$, where α_1 and α_2 either belong to V_N or are split strings. and $A \in V_N$, and $f \in \{C1, C2, W\}$. Definitions of the operations $C1$, $C2$ and W are given above.

The language generated by a MHG G is defined as follows:

$$L(G) = \{ w_1 w_2 \mid S \stackrel{*}{\underset{G}{\Rightarrow}} w_1 \uparrow w_2 \}$$

We now give a MHG generating L_1

$$\begin{array}{ll} S \rightarrow W(S_1, f\uparrow) & S \rightarrow W(S_2, f\uparrow) \\ S_1 \rightarrow C2(a\uparrow, S_3) & S_2 \rightarrow g\uparrow h \\ S_3 \rightarrow W(S_1, b\uparrow c) & S_3 \rightarrow W(S_2, b\uparrow c) \end{array}$$

2.2 Tree Adjoining Grammars and Modified Head Grammars

In this section we show the equivalence of Tree Adjoining Grammars and Modified Head Grammars in their ability to generate string languages. First, we will give an informal discussion about the similarity of the tree adjoining and wrapping operations in the way they combine strings.

2.1.1 Tree Adjunction and Wrapping

Before showing the formal equivalence of MHG's and TAG's, we consider the relationship between the wrapping operation W of MHG's and the adjoining operation of TAG's. Suppose we have the production $p = X \rightarrow W(Y, Z)$ in a MHG G, and that we have two derivations from the nonterminals Y and Z deriving the headed strings $w_1 \uparrow w_2$ and $v_1 \uparrow v_2$ respectively. Given the production p, we can derive the split string $w_1 v_1 \uparrow v_2 w_2$ from X.

Consider the Figure 2.1. Suppose there is a derived auxiliary tree γ in some TAG G, corresponding to the above derivation of $w_1 \uparrow w_2$ from Y where the foot node appears at the split point, as shown in the figure below. Also assume that there is a node η dominating a subtree that corresponds to a derivation of $v_1 \uparrow v_2$ from Z where, as before, we assume that the foot node appears at the split point. Consider the tree resulting from the adjunction of γ at the node η, as shown in the figure. The resulting tree can be thought of as corresponding to the derivation of the split string $w_1 v_1 \uparrow v_2 w_2$ from X. This

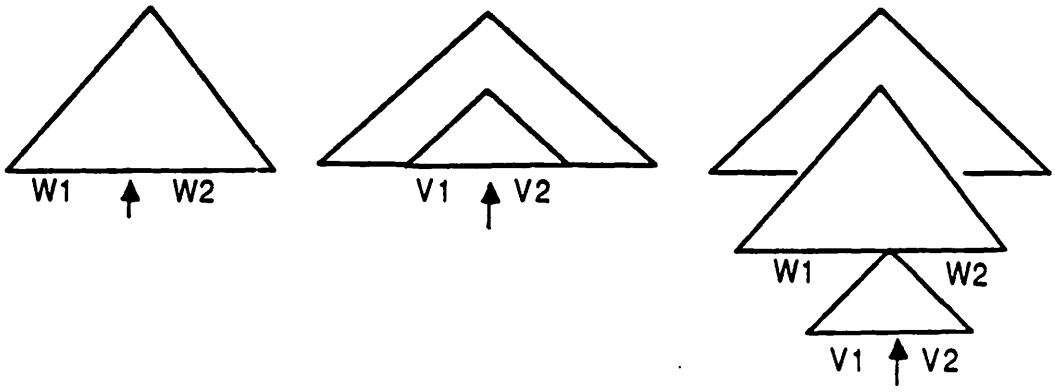

Figure 2.1: Adjunction and wrapping

example illustrates the basic intuition behind the constructions involved in the following proofs showing the equivalence of MHG's and TAG's.

2.2.2 Inclusion of TAL in MHL

Based on the above observation concerning the similarity between the wrapping and adjoining operations, we shall now present an algorithm (*convert*) for transforming a given TAG $G = (V_N, V_T, S, I, A)$ to an equivalent MHG $G' = (V'_N, V_T, S, P)$. In this section, we have generalized the concatentation operations of MHG's to be of the form Cj for $j \geq 1$. These operations can always be simulated using just $C1$ and $C2$.

We shall first describe the algorithm *convert* informally. If *convert* is applied on η, a node of some elementary tree γ, *convert* returns a sequence of productions in the MHG formalism capturing the structure of the subtree of γ rooted at η. The wrapping operation is used to simulate the effect of adjunction, and the concatenation operations $C1$ and $C2$ to concatentate the strings derivable from the left and right children of a node. The choice of either $C1$ or $C2$ depends on which child is the ancestor of the foot node. The exact structure of a tree can be captured by using nonterminals that are named by the addresses of nodes of elementary trees rather than the nonterminals labelling the nodes.

Convert works as follows. Let $\langle \beta, i \rangle$ be the address of a node in an auxiliary tree β, and γ belongs to $\mathcal{D}(\langle \beta, i \rangle)$ with a frontier $w_1 X w_2$. We have a nonterminal corresponding to this node (denoted by $\overline{\langle \beta, i \rangle}$) which derives the split string $w_1 \uparrow w_2$. In particular, when $\langle \beta, i \rangle$ is the root of β (i.e., $i = \epsilon$), then the nonterminal $\overline{\langle \beta, \epsilon \rangle}$ should derive the split strings $w_1 \uparrow w_2$ whenever there is a tree in $\mathcal{D}(\beta)$ with frontier $w_1 X w_2$. That is, the split point appears in a position corresponding to the foot node.

Thus, the wrapping operation W can be used to simulate the effect of adjoining in the following manner. If $\langle \gamma, i \rangle$ is a node at which β is adjoinable, we have a production corresponding to adjunction of β at $\langle \gamma, i \rangle$

$$\overline{\langle \gamma, i \rangle} \rightarrow W(\overline{\langle \beta, \epsilon \rangle}, \underline{\langle \gamma, i \rangle})$$

where $\underline{\langle \gamma, i \rangle}$ derives strings derivable from the children of $\langle \gamma, i \rangle$. We also have the rule

$$\overline{\langle \gamma, i \rangle} \rightarrow \underline{\langle \gamma, i \rangle}$$

for the case when no adjunction takes place at $\langle \gamma, i \rangle$. Since $\underline{\langle \gamma, i \rangle}$ is supposed to derive strings derivable by the concatenation of the frontiers of subtrees dominated by the children of $\langle \gamma, i \rangle$, we have the production,

$$\underline{\langle \gamma, i \rangle} \rightarrow Cj(\overline{\langle \gamma, i \cdot 1 \rangle}, \ldots, \overline{\langle \gamma, i \cdot j \rangle}, \ldots, \overline{\langle \gamma, i \cdot k \rangle})$$

where $\langle \gamma, i \cdot 1 \rangle, \ldots, \langle \gamma, i \cdot j \rangle, \ldots, \langle \gamma, i \cdot k \rangle$ correspond to the k children of $\langle \gamma, i \rangle$ and where the j^{th} child is the ancestor of the foot node. By convention, we let j to be 1 when $\langle \gamma, i \rangle$ itself is not the ancestor of the foot node. We used the operation Cj, since we would like to put the split point in the position where the foot node is.

We are now in a positon to define the conversion process. The algorithm is as follows:

add $S \rightarrow \overline{\langle \beta, \epsilon \rangle}$ for each auxiliary tree adjoinable at the root of the initial tree,
add $S \rightarrow \overline{\lambda}$ if the root of the initial tree does not have an OA constraint.
for each auxiliary tree β, call *convert*($\langle \beta, \epsilon \rangle$)

where the procedure *convert* is as defined below.

define *convert*($\langle \beta, i \rangle$);
case 1: $\langle \beta, i \rangle$ is a leaf node
    if $\langle \beta, i \rangle$ has label $\epsilon \in V_T \cup \{\lambda\}$ then
        add $\overline{\langle \beta, i \rangle} \rightarrow \underline{< \beta, i >}$ to P **(1)**
        add $\underline{< \beta, i >} \rightarrow \epsilon_\dagger$ to P **(2)**
    else { $\overline{\langle \beta, i \rangle}$ *is the foot node* }
        add $\overline{\langle \beta, i \rangle} \rightarrow W(\overline{\langle \beta_1, \epsilon \rangle}, \underline{< \beta, i >})$ **(3)**
            for each β_1 in SA constraint of $\langle \beta, i \rangle$
        add $\overline{\langle \beta, i \rangle} \rightarrow \underline{< \beta, i >}$ **(4)**

if $\langle \beta, i \rangle$ does not have an OA constraint;

add $\underline{< \beta, i >} \rightarrow \varepsilon_{\uparrow}$. (5)

case 2: $\langle \beta, i \rangle$ is an internal node and has k children

add $\overline{\langle \beta, i \rangle} \rightarrow W(\overline{\langle \beta_1, \epsilon \rangle}, \underline{\langle \beta, i \rangle})$ (6)

for each β_1 in SA constraint of $\langle \beta, i \rangle$

add $\overline{\langle \beta, i \rangle} \rightarrow \underline{\langle \beta, i \rangle}$ (7)

if $\langle \beta, i \rangle$ does not have OA constraint;

if $\langle \beta, i \rangle$ is ancestor of foot node then

add $\underline{\langle \beta, i \rangle} \rightarrow Cj(\overline{\langle \beta, i \cdot 1 \rangle}, \ldots . \overline{\langle \beta, i \cdot k \rangle})$ (8)

where j^{th} child dominates foot node;

else add $\underline{\langle \beta, i \rangle} \rightarrow C1(\overline{\langle \beta, i \cdot 1 \rangle}, \ldots, \overline{\langle \beta, i \cdot k \rangle})$ (9)

for $1 \leq j \leq k$ do *convert*$(\langle \beta, i \cdot j \rangle)$.

Lemma 2.1 $L(G) \subseteq L(G')$

To prove the inclusion of $L(G)$ in $L(G')$, we induct on the height of the derivation trees characterizing trees derived from auxiliary trees and on the addresses of nodes in these auxiliary trees to prove the following proposition.

Proposition 2.1 For all auxiliary trees, β, let $\Upsilon \in T_D(G)$ be of height h_1, describing the derivation of $\gamma \in D(\beta)$. Let the node denoted by $\langle \beta, i_1 \rangle$ have k_{i_1} children. Let $\gamma_{i_1} \in D(\langle \beta, i_1 \rangle)$ and $\gamma_{i_1 j} \in D(\langle \beta, i_1 j \rangle)$ for $1 \leq j \leq k_1$ be described by Υ. Then one of the following two cases holds.

- Let the node denoted by $\langle \beta, i_1 \rangle$ dominate the foot node (which is labelled by X say). If the frontier of γ_{i_1} is $w_1 X w_2$, and the concatenation the frontiers of the trees $\gamma_{i_1 j}$ for $1 \leq j \leq k_{i_1}$ be $w_3 X w_4$, then

  - $\overline{<\beta, i_1>} \underset{G'}{\overset{*}{\Longrightarrow}} w_{1\uparrow} w_2$
  - $\underline{< \beta, i_1 >} \underset{G'}{\overset{*}{\Longrightarrow}} w_{3\uparrow} w_4$

- Let the node denoted by $\langle \beta, i_1 \rangle$ not dominate the foot node If the frontier of γ_{i_1} is w_1, and the concatenation the frontiers of the trees $\gamma_{i_1 j}$ for $1 \leq j \leq k_{i_1}$ be w_2, then

  - $\overline{<\beta, i_1>} \underset{G'}{\overset{*}{\Longrightarrow}} \overline{w_1}$
  - $\underline{<\beta, i_1>} \underset{G'}{\overset{*}{\Longrightarrow}} \overline{w_2}$

The base case corresponds to considering the frontier node of β, i.e., height of the derivation tree is one. By the productions introduced by step 1 and 2, the base case holds.

For the inductive step, consider $\gamma_i \in \mathcal{D}(\langle \beta, i \rangle)$ whose derivation is represented by Υ of height h. Let the inductive hypothesis hold for all $h_1 < h$ and $i_1 < i$. We will have the production

$$\underline{<\beta, i>} \rightarrow C_j(\overline{<\beta, 11>} \ldots \overline{<\beta, im>}$$

and for $1 \leq k \leq m$, $\langle \beta, ik \rangle$ satisfies the inductive hypothesis condition. Thus, $\underline{<\beta, i>}$ will satisfy the inductive hypothesis. In considering $\overline{<\beta, i>}$, we have to take into account two cases. If according to Υ, no adjunction takes place at $\langle \beta, i \rangle$, then since there is a production

$$\overline{<\beta, i>} \rightarrow \underline{<\beta, i>}$$

$\overline{<\beta, i>}$ derives the string desired. If on the other hand, adjunction takes place at $\langle \beta, i \rangle$ according to the Υ which has the form shown below.

Then the height of Υ_1 is less than h. We have already seen that inductive hypothesis will hold for $\underline{<\beta, i>}$. Since G' has the production

$$\overline{<\beta, i>} \rightarrow W(\overline{<\beta_1, \epsilon>}, \underline{<\beta, i>})$$

and the fact that the induction hypothesis will hold for $\overline{<\beta_1, \epsilon>}$, given that wrapping and adjunction have the same effect on strings, we have shown that the inductive hypothesis

holds for $\overline{<\beta, i>}$ as well. When we reach the root of the tree, i.e., $i = \epsilon$, then we would have shown the inductive hypothesis holds for the derivation trees of height h also.

To show that $L(G) \subseteq L(G')$, we have to just consider the rules
$S \rightarrow \overline{\langle \beta, \epsilon \rangle}$ for each auxiliary tree adjoinable at the root of the initial tree,
add $S \rightarrow \overline{\lambda}$ if the root of the initial tree does not have an OA constraint.

Lemma 2.2 $L(G') \subseteq L(G)$

We prove this result by proving by induction the following proposition.

Proposition 2.2 Assume that a node $\langle \beta, i \rangle$ in an auxiliary tree β has k_i children. If $\overline{<\beta, i>} \stackrel{k}{\underset{G'}{\Longrightarrow}} w_1 \downarrow w_2$ then there is a $\gamma \in D(\langle \beta, i \rangle)$ whose frontier is $w_1 X w_2$ (if the node $\langle \beta, i \rangle$ dominates the foot node whose label is X) or $w_1 w_2$ (if this node does not dominate the foot node).
If $\underline{<\beta, i>} \stackrel{k}{\underset{G'}{\Longrightarrow}} w_1 \downarrow w_2$ then there are $\gamma_j \in D(\langle \beta, ij \rangle)$ for $1 \leq j \leq k_i$ which are described by the same derivation tree Υ such that the concatenation of the frontier of these trees is $w_1 X w_2$ (if the node $\langle \beta, i \rangle$ dominates the foot node whose label is X) or $w_1 w_2$ (if this node does not dominate the foot node).

The base case corresponds to the node $\langle \beta, i \rangle$ being a leaf node, and the hypothesis holds because of the productions given in step 1. For the inductive case, we assume that for all $k < m$ the above proposition holds. If $\underline{<\beta, i>} \stackrel{m}{\underset{G'}{\Longrightarrow}} w_1 \downarrow w_2$ then the last production used is that given in step 8 or 9. We will then have to consider the derivations from $\overline{<\beta, ij>}$ for $1 \leq j \leq k_i$, each of which has to be less than m steps. Thus, the inductive step holds for these derivations and using the productions in steps 8 or 9, we are assured the hypothesis holds for the m step derivation too. If $\overline{<\beta, i>} \stackrel{m}{\underset{G'}{\Longrightarrow}} w_1 \downarrow w_2$ then the the last step used must have been that given by 6 or 7. In either case we can be guaranteed that the hypothesis holds for this m step derivation too.

As an illustration of the conversion process, we will consider the TAG (whose elementary trees are given in Figure 2.2) generating the language L_1 given in Section 2.1.2

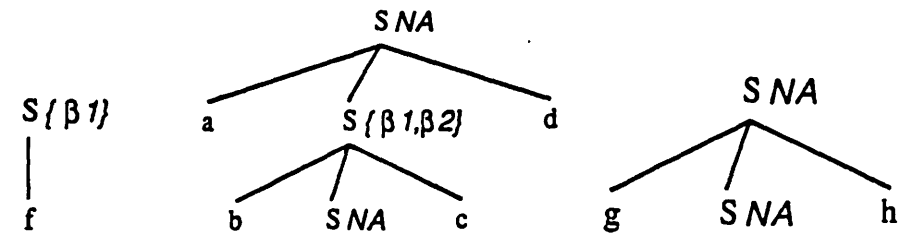

Figure 2.2: A TAG which generates L_1

can be converted into a Modified Head Grammar that generates this language.

$S \rightarrow \overline{\langle \alpha, \epsilon \rangle}$

$\overline{\langle \alpha, \epsilon \rangle} \rightarrow W(\overline{\langle \beta_1, \epsilon \rangle}, \underline{\langle \alpha, \epsilon \rangle})$ $\qquad$ $\overline{\langle \alpha, \epsilon \rangle} \rightarrow W(\overline{\langle \beta_2, \epsilon \rangle}, \underline{\langle \alpha, \epsilon \rangle})$

$\underline{\langle \alpha, \epsilon \rangle} \rightarrow \overline{\langle \alpha, 1 \rangle}$

$\overline{\langle \alpha, 1 \rangle} \rightarrow f_\uparrow$

$\overline{\langle \beta_1, \epsilon \rangle} \rightarrow \underline{\langle \beta_1, \epsilon \rangle}$

$\underline{\langle \beta_1, \epsilon \rangle} \rightarrow C2(\overline{\langle \beta_1, 1 \rangle}, \overline{\langle \beta_1, 2 \rangle})$

$\overline{\langle \beta_1, 1 \rangle} \rightarrow a_\uparrow$

$\overline{\langle \beta_1, 2 \rangle} \rightarrow W(\overline{\langle \beta_1, \epsilon \rangle}, \underline{\langle \beta_1, 2 \rangle})$ $\qquad$ $\overline{\langle \beta_1, 2 \rangle} \rightarrow W(\overline{\langle \beta_2, \epsilon \rangle}, \underline{\langle \beta_1, 2 \rangle})$

$\underline{\langle \beta_1, 2 \rangle} \rightarrow C2(\overline{\langle \beta_1, 2 \cdot 1 \rangle}, \overline{\langle \beta_1, 2 \cdot 2 \rangle}, \overline{\langle \beta_1, 2 \cdot 3 \rangle})$

$\overline{\langle \beta_1, 2 \cdot 1 \rangle} \rightarrow b_\uparrow$

$\overline{\langle \beta_1, 2 \cdot 3 \rangle} \rightarrow c_\uparrow$

$\overline{\langle \beta_1, 2 \cdot 2 \rangle} \rightarrow \overline{\lambda}$

$\overline{\langle \beta_2, \epsilon \rangle} \rightarrow \underline{\langle \beta_2, \epsilon \rangle}$

$\underline{\langle \beta_2, \epsilon \rangle} \rightarrow C2(\overline{\langle \beta_2, 1 \rangle}, \overline{\langle \beta_2, 2 \rangle}, \overline{\langle \beta_2, 3 \rangle})$

$\overline{\langle \beta_2, 1 \rangle} \rightarrow g_\uparrow$

$\overline{\langle \beta_2, 3 \rangle} \rightarrow h_\uparrow$

$\overline{\langle \beta_2, 2 \rangle} \rightarrow \overline{\lambda}$

We can simplify this grammar, replacing $\overline{\langle \beta_1, \epsilon \rangle}$ by S_1; $\overline{\langle \beta_2, \epsilon \rangle}$ by S_2; and other such minor renaming to produce exactly the same grammar for L_1 as given in Section 2.1.2.

2.2.3 Inclusion of MHL in TAL

When we convert a TAG into an MHG, each elementary tree generates a set of productions. The sets generated by any two distinct elementary trees are disjoint and, furthermore, have a constrained form encoding the hierarchical structure of the tree. The task of converting an MHG to a TAG cannot simply involve the inversion of this construction since it is not in general possible to find groupings of productions in an MHG that have the required structure.

The approach used to convert MHG's to TAG's is based on satisfying the following requirement: for each derivation in the MHG there must be a derived tree in the TAG for the same string, in which the foot is positioned at the split point. The first problem that will be addressed is representing derivations in a MHG as a derived tree in a TAG. We can describe the relationship between these two forms of representation by induction on the length of the derivation. The derivations

$$A \overset{*}{\underset{G}{\Rightarrow}} {}_\uparrow\varepsilon \qquad \text{or} \qquad A \overset{*}{\underset{G}{\Rightarrow}} \varepsilon_\uparrow$$

using the productions

$$A \rightarrow {}_\uparrow\varepsilon \qquad \text{or} \qquad A \rightarrow \varepsilon_\uparrow$$

have the following associated derived trees:

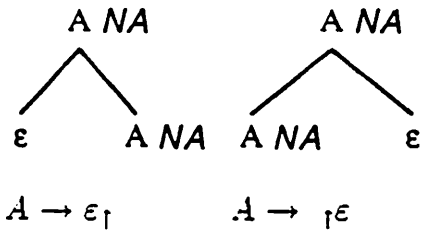

Figure 2.3: Derived trees for terminal productions

Given derived trees corresponding to derivations for B and C, use of the productions

$$A \rightarrow C1(B, C) \qquad \text{or} \qquad A \rightarrow C2(B, C)$$

would generate the following trees representing derivations from A:

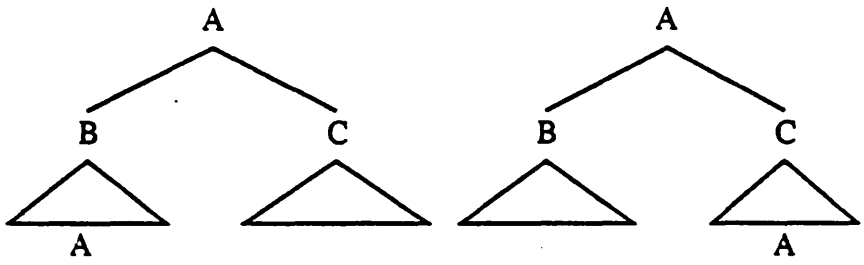

Figure 2.4: Derived trees for C_1 and C_2 productions

Given a derivation tree for C, use of the production:

$$A \rightarrow W(B, C)$$

would generate the following tree representing a partial derivation from A into which a derived tree for B must be adjoined before the tree is complete. We use an OA constraint to ensure that the tree is completed.

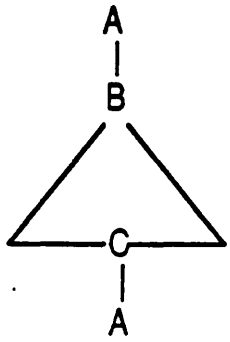

Figure 2.5: Derived trees for W productions

Notice that we are positioning the foot node in these derived trees in such a way as to ensure that the string is split in the correct place.

We now discuss how to construct a TAG that generate all the derived trees mentioned above. In other words we should establish the following proposition:

Proposition 2.3

In the TAG G there is a derived auxiliary tree γ having no OA constraints, with root labelled A and frontier $w_1 A w_2$ if and only if $A \overset{*}{\underset{G}{\Longrightarrow}} w_1 \downarrow w_2$.

Given an MHG $G = (V_N, V_T, S, P)$, we show that there exists an equivalent TAG $G' = (V'_N, V_T, S, I, A)$. Without loss of generality, we assume a normal form that uses productions of the following form:

$$A \rightarrow f(B, C) \quad \text{or} \quad A \rightarrow \downarrow\varepsilon \quad \text{or} \quad A \rightarrow \varepsilon\downarrow$$

where $A, B, C \in V_N$, $\varepsilon \in V_T \cup \{\lambda\}$ and $f \in \{C1, C2, W\}$. The conversion proceeds as follows:

1. If $A \rightarrow \downarrow\varepsilon \in P$ then

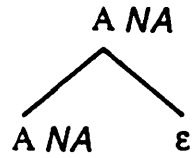

Figure 2.6: $A \rightarrow \downarrow\varepsilon$

2. If $A \rightarrow \varepsilon\downarrow \in P$ then

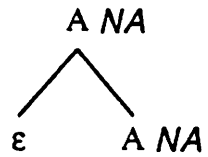

Figure 2.7: $A \rightarrow \varepsilon\downarrow$

3. If $A \rightarrow C1(B, C) \in P$ then

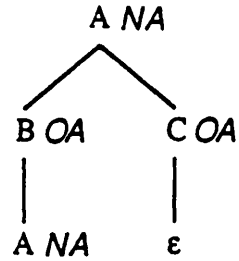

Figure 2.8: $A \rightarrow C1(B, C)$

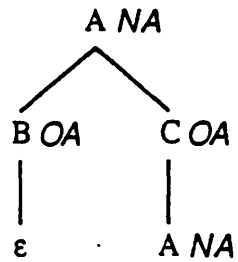

Figure 2.9: $A \rightarrow C2(B, C)$

4. If $A \rightarrow C2(B, C) \in P$ then

5. If $A \rightarrow W(B, C) \in P$ then

The set I of initial trees consists of the single tree α:

We have used OA constraints to ensure that nonterminals introduced by the MHG productions are rewritten and NA constraints to ensure that once a nonterminal is rewritten, it is not used again. We prove the equivalence of $L(G)$ and $L(G')$ by showing that Proposition 2.3 is satisfied.

Lemma 2.3 $L(G) \subseteq L(G')$

To prove this lemma we have to show that if $X \in V_N$ and $X \overset{*}{\underset{G}{\Longrightarrow}} w_1 w_2$ then there is a derived tree with no OA constraints as shown below:

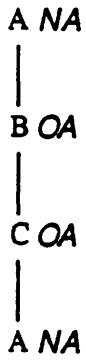

Figure 2.10: $A \rightarrow W(B, C)$

Figure 2.11: Initial tree

We carry out the induction on the length of derivations of strings in G. Suppose we assume that if $B \stackrel{i}{\underset{G}{\Longrightarrow}} u_1 \downarrow u_2$ and $C \stackrel{j}{\underset{G}{\Longrightarrow}} v_1 \downarrow v_2$ then there are derived auxiliary trees γ_1 and γ_2 as shown in Figure 2.13. We must consider each type of production in P.

For example, if the production $A \rightarrow C1(B, C) \in P$ then $A \stackrel{k}{\underset{G}{\Longrightarrow}} u_1 \downarrow u_2 v_1 v_2$ where $k = i + j + 1$. Since $A \rightarrow C1(B, C) \in P$ we have the auxiliary tree β. Adjoining γ_1 and γ_2 into β we obtain the required derived auxiliary tree γ shown in Figure 2.13.

The other cases can be handled in a similar manner to show that lemma 1 holds.

Lemma 2.4 $L(G') \subseteq L(G)$

We show by induction that

For all $\gamma \in \mathcal{D}(\beta)$ having no OA constraints where the frontier of γ is

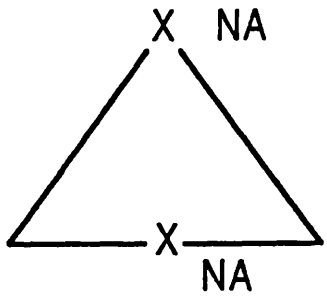

Figure 2.12: Proposition for MHG to TAG

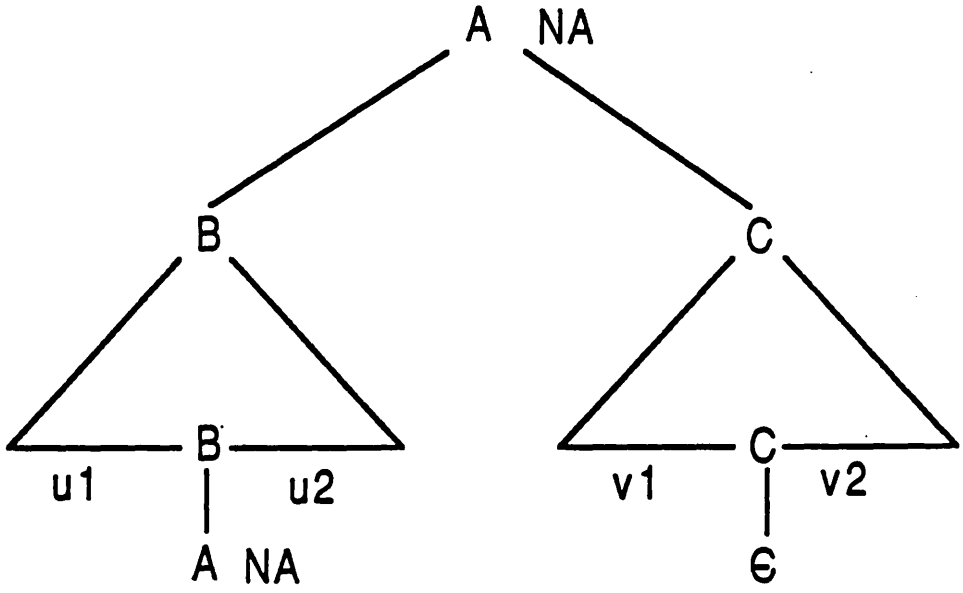

Figure 2.13: TAG derivation corresponding to use of $A \rightarrow C1(B, C)$

w_1Xw_2 and the height of the derivation tree Υ, corresponding to a derivation of γ, has a height k then $X \stackrel{*}{\underset{G}{\Longrightarrow}} w_1w_2$.

For the inductive step, we assume that the above statement holds for all derivation trees of height $\leq k$. Consider a derived auxiliary tree $\gamma \in \mathcal{D}(\beta)$ whose frontier is w_1Aw_2. Let the derivation tree Υ corresponding to the derivation of γ have height $k+1$. Since $\gamma \in \mathcal{D}(\beta)$, the root of Υ is labelled by β and Υ is of the form shown in Figure 2.14.

There are several cases to be considered. If A was introduced because of the production $A \rightarrow C1(B, C) \in P$ and if u_1Bu_2 and v_1Cv_2 are the frontiers of the trees that are

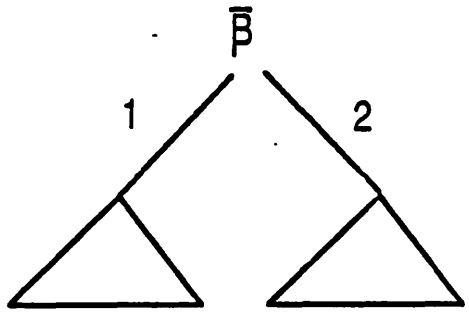

Figure 2.14: Derivation tree for γ

adjoined at the nodes labelled B and C respectively, then $w_1 = u_1$ and $w_2 = u_2 v_1 v_2$. Since the height of γ_1 and γ_2 are $\leq k$ by the inductive hypothesis:

$$B \underset{G}{\overset{*}{\Longrightarrow}} u_1 \downarrow u_2 \qquad \text{and} \qquad C \underset{G}{\overset{*}{\Longrightarrow}} v_1 \downarrow v_2$$

Since $A \rightarrow C1(B.C) \in P$,

$$A \underset{G}{\overset{*}{\Longrightarrow}} u_1 \downarrow u_2 v_1 v_2 = w_1 \downarrow w_2$$

In a similar way, we can deal with each of the other possible auxiliary trees β that could label the root of Υ. We can then conclude that by the above induction lemma 2 holds; and hence by lemma 1 and lemma 2 we show $L(G) = L(G')$.

2.2.4 Comments on the Relationship between Tree Adjoining Languages and Head Languages

In this section, we established that the two grammtical formalisms, Tree Adjoining Grammars and Modified Head Grammars, generate the same class of languages. [jvw86,vwj86] discusses the relationship between MHG's and HG's and describe a proof showing the inclusion of Head Languages (HL) in TAL. The remaining inclusion of MHL in HL, that would demonstrate the equivalence of HG's and MHG's, has not yet been established in the general case. It has been shown in [Joshi 86,VijayShanker 86b] that the relationship

between HG's and MHG's is very close, the differences hinging on the status of heads of empty strings and whether one deals with the heads directly or with the left and right wrapping positions around the head. Aside from this minor notational variation, there is not much difference in the class of languages generated. A linguistic view of the relationship between TAG's and HG's has been discussed in [Weir 86], where the question of the strong generative capacity of the two formalisms and their ability to give certain linguistic analyses has been addressed.

The inclusion relationship that exists between the three formalisms is pictorially shown below.

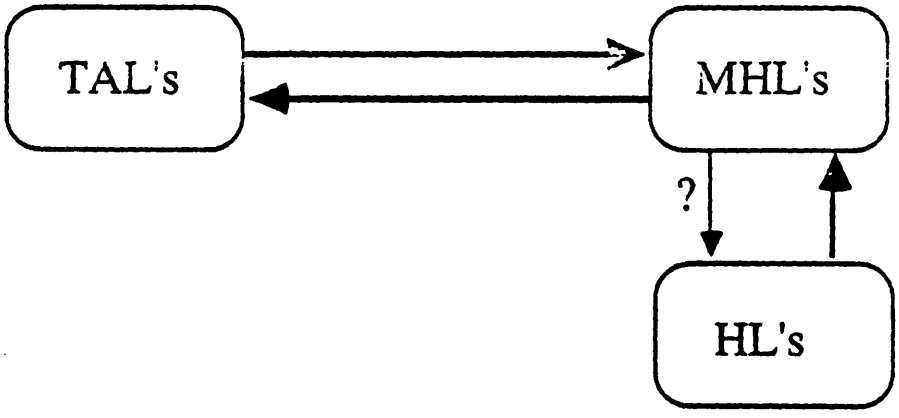

Figure 2.15: Summary of the relationship between TAG, MHG, and HG

Chapter 3

Automata for TAL's

In this chapter, we continue the study of Tree Adjoining Grammars by giving automata characterization for the class of languages generated by them. This characterization helps us understand the TAG formalism and its ability to generate languages. We make comparisons with corresponding automata for Context Free Grammars and Indexed Grammars, so that we can see how TAG's fit between them. In Section 3.1, we define a class of string automata, called the *Embedded Pushdown Automata*, and show that they are equivalent to TAG's. We then prove two results about TAL's using epda's, its equivalence to a linear version of Indexed Grammars and closure of TAL's under intersection with regular languages.

3.1 Embedded Pushdown Automaton

In this section we define a class of automata, called the embedded pushdown automata (epda), and show that the class of languages they recognize is exactly TAL's. An epda is an extension of the pda and may be thought of as a second order pushdown *acceptor.*

A epda consists of a finite state control, a one way input tape, and a store that is made up of a pushdown of nonempty pushdown stores containing stack symbols. Like the pda, the stack pointer always points to the topmost element in the store; thus, in the case of epda, the stack pointer points to the top element of the top stack[1].

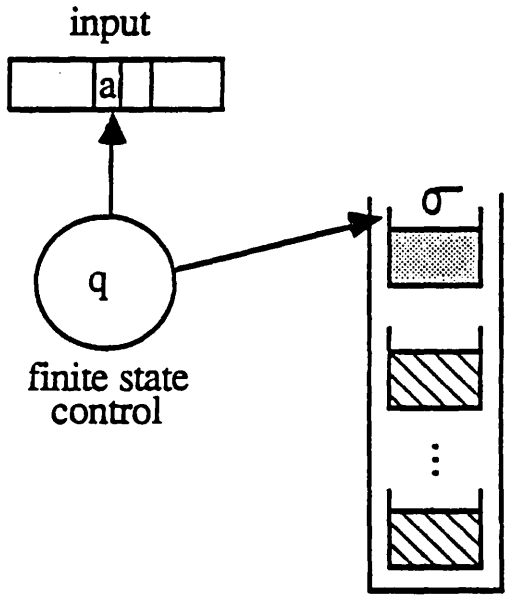

Figure 3.1: Embedded pushdown automaton

A move of a pda may be described as replacing the top element of the stack by a finite sequence (possibly empty) of stack symbols based on the input, the state of the finite state control, and the top stack symbol. After this move takes place, the stack pointer moves to point to the new top of the stack. A move of an epda may be depicted in a similar way, except that the move of this second order pushdown recognizer has to be explained in two stages, one for each of the two orders of stack. As in the case of pda, based on the input, the state, and the top element, σ, of the the top stack, Υ, the top element is replaced by a finite sequence (possibly empty) of stack elements to give a new top stack Υ'. This resulting pushdown Υ' is then replaced by a sequence of k,

[1]Though the epda has a nested stack store like the nested stack automata (the nested stack automata (nsa) was introduced by Aho [Aho 69], a one way nondeterministic nsa characterizes the class of Indexed languages), it differs from the nsa in that the stack pointer is always at the topmost position.

($k \geq 0$) pushdowns (which includes Υ' if it is nonempty). This move is illustrated in Figure 3.2, and is expressed by defining the transition function, δ, of the epda as follows. In the notation we use, the top of the stack is on the right.

$$(p, \ddagger\sigma'_k\ddagger \ldots \ddagger\sigma'_{i+1}, \sigma_n \ldots \sigma_1, \ddagger\sigma'_i\ddagger \ldots \ddagger\sigma'_1) \in \delta(q, a, \sigma)$$

The marker, $\ddagger$, does not belong to Γ, the set of stack symbols. It is used to separate strings of stack symbols belonging to different stacks. This move therefore replaces the top symbol, σ, of the top stack, Υ, by $\sigma_1 \ldots \sigma_n$, to give a new stack Υ'. New stacks, Υ_j, containing σ'_j for $i+1 \leq j \leq n$ above Υ' and new stacks given by $\ddagger\sigma'_j$ for $1 \leq j \leq i$ immediately below Υ'. As in the pda, the stack pointer moves to the topmost position, i.e., it now points at the stack element given by σ_{n,l_n}.

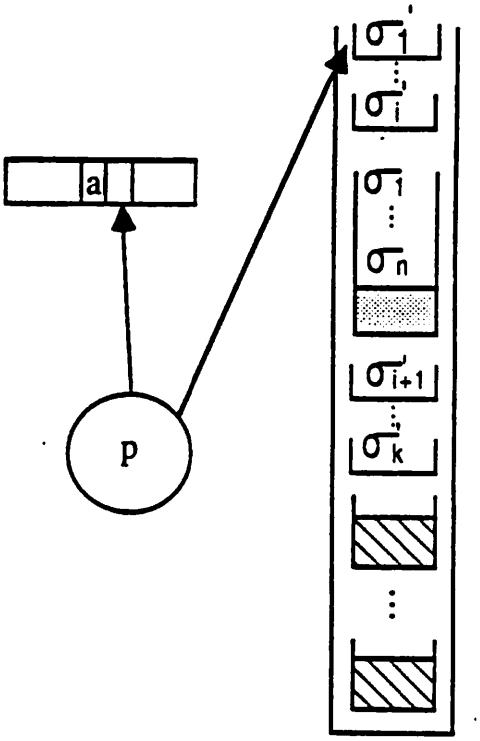

Figure 3.2: Epda after a move

When the top stack empties, the stack pointer automatically moves to the top of the

stack below it. Acceptance by an epda can be defined in terms of empty store or final state. As in the case of the pda, we shall show that the two definitions are equivalent.

We are now in a position to formally define an epda.

Definition 3.1 An epda, M, is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Q_F, \sigma_0)$ where
Q is a finite set of states
$Q_F \subseteq Q$ is the set of final states
Σ is finite set of input symbols
Γ is a finite set of stack symbols
$q_0 \in Q$ is the start state
σ_0 is the symbol on the stack initially
δ is the transition function mapping $Q \times \Sigma \times \Gamma$ to finite subsets of $Q \times \Upsilon^* \times \Gamma^* \times \Upsilon^*$ where Υ correspond to pushdowns of stack symbols and hence are members of Γ^*.

Definition 3.2 Instantaneous Descriptions (ID)

The snapshot of an epda is given by the instantaneous description. An ID is of type $Q \times \Upsilon^* \times \Sigma^* \times \Sigma^*$. An example of an instantaneous description (also called a configuration) is given by:

$$\langle q, \ddagger\sigma_{1,1} \ldots \sigma_{1,l_1} \ddagger \ldots \ddagger\sigma_{i-1,1} \ldots \sigma_{i-1,l_{i-1}} \ddagger\sigma_{i,1} \ldots \sigma_{i,l_i}, w_1, w_2 \rangle$$

where $q \in Q$; w_1 is the initial part of the input that has already been consumed by the epda; w_2 is the part of the input that is yet to be read, with the head positioned on the leftmost symbol of w_2; and $\ddagger\sigma_{1,1} \ldots \sigma_{1,l_1} \ddagger \ldots \ddagger\sigma_{i,1} \ldots \sigma_{i,l_i}$ is the store configuration where $\ddagger\sigma_{i,1} \ldots \sigma_{i,l_i}$ is the topmost stack with the σ_{i,l_i} at the top. The starting ID is given by $\langle q_0, \ddagger\sigma_0, \epsilon, w \rangle$ where w is the input to be read.

Definition 3.3 Language accepted by epda, $M = (Q, \Sigma, \Gamma, \delta, q_0, Q_F, \sigma_0)$, by empty store: $N(M)$.

$$N(M) = \{w \mid \langle q_0, \ddagger\sigma_0, \epsilon, w\rangle \models \langle q, \epsilon, w, \epsilon\rangle \text{ for some } q \in Q.$$

Definition 3.4 Language accepted by epda, $M = (Q, \Sigma, \Gamma, \delta, q_0, Q_F, \sigma_0)$, by final state: $L(M)$.

$L(M) = \{w \mid \langle q_0, \ddagger\sigma_0, \epsilon, w\rangle \models \langle q_f, \Upsilon_1 \ldots \Upsilon_n, w, \epsilon\rangle$ for some $q_f \in Q_F$ and some store configuration $\Upsilon_1 \ldots \Upsilon_n$.

We now give a normal form for epda's. In this normal form, every new stack added in a move has just one stack element. The number of elements that can be added to the top stack (before the move) is at most two.

Theorem 3.1

For every epda, M, we can effectively find an epda, M', such that $N(M) = N(M')$ and $L(M) = L(M')$, where the moves of M' have the following form. For $\varepsilon \in \Sigma \cup \{\epsilon\}$, we have

$$\langle p, \ddagger\sigma_1\ddagger \ldots \ddagger\sigma_i, \sigma_1 \ldots \sigma_n, \ddagger\sigma_{i+1}\ddagger \ldots \ddagger\sigma_k\rangle \in \delta'(q, \varepsilon, \sigma)$$

i.e., the size of each new stack added is one , and number of elements added to the stack which was on top is at most two, i.e., $n \leq 2$.

very badly stated!

We now give the construction of M'. For each move in M of the form

$$\langle p, \ddagger\sigma_{1,1} \ldots \sigma_{1,l_1}\ddagger \ldots \ddagger\sigma_{i,1} \ldots \sigma_{i,l_i}, \sigma_1 \ldots \sigma_n,$$
$$\ddagger\sigma_{i+1,1} \ldots \sigma_{i+1,l_{i+1}}\ddagger \ldots \ddagger\sigma_{k,1} \ldots \sigma_{k,l_k}\rangle \in \delta(q, \varepsilon, \sigma)$$

M' has the following moves.

$\langle p, \ddagger\sigma'_{1,0}\ddagger \ldots \ddagger\sigma'_{i,0}, \sigma^{0,n}, \ddagger\sigma'_{i+1,0}\ddagger \ldots \ddagger\sigma'_{k,0}\rangle \in \delta(q, \varepsilon, \sigma)$

$\langle r, \epsilon, \sigma_{j,i}\sigma'_{j,i+1}, \epsilon\rangle \in \delta(r, \epsilon, \sigma'_{j,i}) \quad \forall r \in Q, 1 \leq j \leq k, 0 \leq i \leq lj - 1$

$\langle r, \epsilon, \sigma_i\sigma^{i+1,n}, \epsilon\rangle \in \delta(r, \epsilon, \sigma^{i,n}) \quad \forall r \in Q) \leq i \leq n$

As an example, we give the transition function of the epda, M, which recognizes the language $\{a^n b^n c^n d^n \mid n \geq 0\}$ by empty store.

$$\delta(q_0, a, \sigma_0) = \{(q_0, \ddagger D, B, \epsilon)\}$$
$$\delta(q_0, a, B) = \{(q_0, \ddagger D, BB, \epsilon)\}$$
$$\delta(q_0, b, B) = \{(q_1, \ddagger C, \epsilon, \epsilon)\}$$
$$\delta(q_1, b, B) = \{(q_1, \ddagger C, \epsilon, \epsilon)\}$$
$$\delta(q_1, c, C) = \{(q_2, \epsilon, \epsilon, \epsilon)\}$$
$$\delta(q_2, c, C) = \{(q_2, \epsilon, \epsilon, \epsilon)\}$$
$$\delta(q_2, d, D) = \{(q_3, \epsilon, \epsilon, \epsilon)\}$$
$$\delta(q_3, d, D) = \{(q_3, \epsilon, \epsilon, \epsilon)\}$$
$$\delta(q_0, \epsilon, \sigma_0) = \{(q_0, \epsilon, \epsilon, \epsilon)\}$$

The automata works as follows. When it encounters the initial a's it pushes in B's in the top stack so that it can check that the number of a's and b's are the same. At the same time it ensures that the number of a's and d's are the same by pushing new stacks, with just the symbol D on them, below the top most stack. The number of b's and c's are checked in the same manner.

Theorem 3.2 For every epda, M, we can effectively find an epda, M', such that $L(M) = N(M')$.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Q_F, \sigma_0)$. M' is defined by $(Q \cup \{q'\}, \Sigma, \Gamma, \delta', q_0, Q_F, \sigma_0)$ where $q' \notin Q$. δ' is given by adding to δ the following moves.

$$\langle q', \epsilon, \epsilon, \epsilon \rangle \in \delta'(q_f, \epsilon, \sigma), \forall \sigma \in \Gamma, \forall q_f \in Q_F \cup \{q'\}.$$

M' simulates M except that when M reaches a final state, M' can nondeterministically choose to empty its stack. Clearly $N(M') = L(M)$ as required.

Theorem 3.3 For every epda, M, there is an epda M', such that $N(M) = L(M')$.

Let M be defined as $(Q, \Sigma \ \Gamma, \delta, q_0, Q_F, \sigma_0)$. Then $M' = (Q \cup \{q_0'\}, \Sigma, \Gamma \cup \{\sigma_0'\}, \delta', q_0', \{q'\}, \sigma_0')$ where $q', q_0' \notin Q, \sigma \notin \Gamma$. M' works as follows. It has a new initial stack

symbol, σ'_0, on top of which M' immediately adds the initial stack symbol of M. Then, M' uses the same moves as M. When M reaches the empty store configuration, M' will have only one stack which contains just σ'_0. M' can then choose to go to the final state q'_f (from which there are no moves defined) and accept the string. However, if M has not completely consumed the input, then M can continue to recognize the rest of the string if there are moves defined which do not look at the top of the stack (since the stack is empty), i.e., independent of the symbol pointed to by the stack pointer. This move can be simulated by M' in the following way. We can not let any new stacks be inserted below the top stack if the top stack has σ'_0 at its top (since this stack has always got to be at the bottom). Thus, the move is simulated by checking if the top stack symbol is an element of Γ, and if it is then to perform exactly as M. But if the top stack symbol is σ'_0 then M' makes a similar move except to insert a stack containing just σ'_0 at the bottom. Thus, δ' is defined as follows. In addition to the moves given in δ, δ' contains the moves

$$\langle q_0, \ddagger\sigma'_0, \sigma_0, \epsilon\rangle \in \delta'(q'_0, \epsilon, \sigma'_0)$$
$$\langle q', \epsilon, \epsilon, \epsilon\rangle \in \delta'(q, \epsilon, \sigma'_0) \quad \forall q \in Q$$

If δ is defined such that $\langle p, \ddagger\sigma_1\ddagger \ldots \ddagger\sigma_i, \sigma_1\sigma_2, \ddagger\sigma_{i+1}\ddagger \ldots \ddagger\sigma_k\rangle \in \delta(q, \varepsilon, \epsilon)$ for $\varepsilon \in \Sigma \cup \{\epsilon\}$ then δ' has the moves:

$$\langle p, \ddagger\sigma_1\ddagger \ldots \ddagger\sigma_i, \sigma_1\sigma_2, \ddagger\sigma_{i+1}\ddagger \ldots \ddagger\sigma_k\rangle \in \delta'(q, \varepsilon, \sigma) \quad \forall\sigma \in \Gamma$$
$$\langle p, \ddagger\sigma'_0\ddagger\sigma_1\ddagger \ldots \ddagger\sigma_i, \sigma_1\sigma_2, \ddagger\sigma_{i+1}\ddagger \ldots \ddagger\sigma_k\rangle \in \delta'(q, \varepsilon, \sigma'_0)$$

3.2 TAG's and epda's

In this section, we show that TAG's and epda are equivalent, i.e., every TAL is recognized by an epda, and every language recognized by an epda is a TAL. In Section 3.2.1, we informally discuss the relationship between TAG's and epda's.

3.2.1 Informal Discussion on TAG's and epda's

We now informally discuss the equivalence of epda's and TAG's. We can relate the moves of an epda to the expansion of nodes in a top down left-to-right order derivation in a TAG. When an epda imitates a TAG derivation, a stack is associated with each node as shown in Figure 3.3.

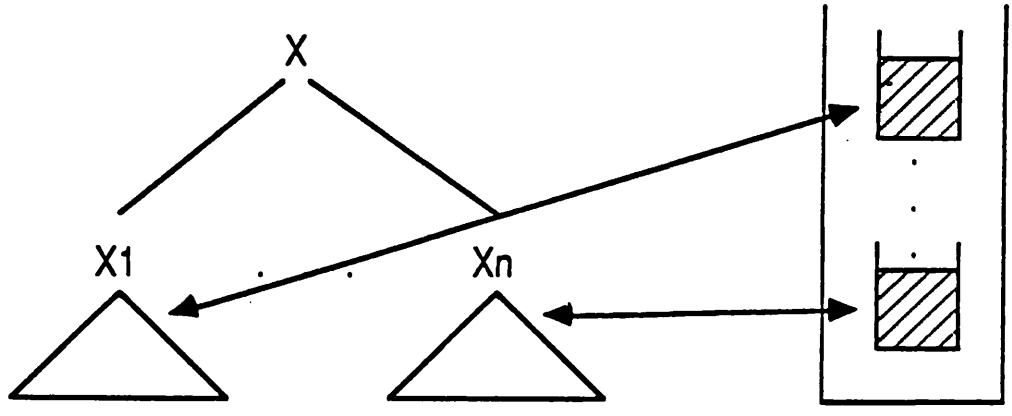

Figure 3.3: Association of subtrees of TAG with stacks in epda

The stack associated with the node to be expanded is assumed to be at the top. If we adjoin at this node, then new stacks are placed above and below the present top stack encoding the parts of the auxiliary tree to the left and right of its spine. Thus, the epda will recognize the left part of the auxiliary tree, expand the subtree below the node where adjunction took place, before recognizing the string to the right of the foot node of the auxiliary tree. This correspondence is illustrated by the Figure 3.4 below.

We can give constructions from a TAG to an epda and vice versa (based on the above discussion) to establish the equivalence of the classes of languages generated by TAG's and those recognized by epda's. However, the proofs involving TAG's are long and tedious, involving two inductions for the structure of the derivation tree and the structure of the elementary trees. Note that there is a similarity in our discussions about the correspondence between TAG's and epda's (given above) and the correspondence

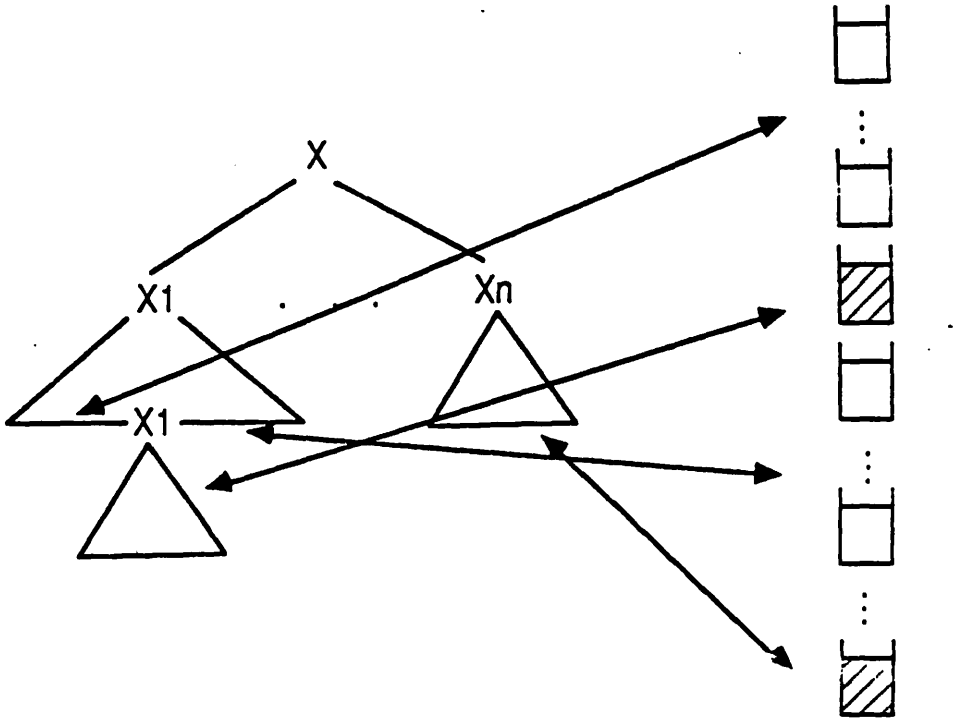

Figure 3.4: Move corresponding to adjunction

between TAG's and MHG's. We give constructions showing the equivalence between MHG's and epda's and having already established the equivalence between MHG's and TAG's, we can then conclude that TAG's and epda's are equivalent.

The main idea behind the equivalence of epda's and MHG's is as follows. Suppose we use the production

$$A \rightarrow W(B, C)$$

and suppose $B \stackrel{*}{\Longrightarrow} u_1 \uparrow u_2$. Then A will derive a string in which $u_1 \uparrow u_2$ wraps around the string derived by C. In order for an epda to recognize this string, we assume that the top stack is $\ddagger CB$. Seeing B at the top of the store, the epda can insert stacks $\Upsilon_{k+1} \ldots \Upsilon_n$ above and the stacks $\Upsilon_1 \ldots \Upsilon_k$ below this stack. Thus the stacks on top of the store of the epda is given by $\Upsilon_1 \ldots \Upsilon_k \ddagger C \Upsilon_{k+1} \ldots \Upsilon_n$ (see the figure below). The stacks above will be emptied while reading the input u_1; the epda will then work to accept the string derived by C (note that even if new stacks are introduced below this stack because of the symbol C on the top of the store, they will be emptied before the stacks $\Upsilon_1 \ldots \Upsilon_k$,

introduced because of B. At the top of the store now are the stacks $\Upsilon_1 \ldots \Upsilon_k$, which will be emptied when the epda scans the input u_2. Thus, the epda simulates the derivation from A of the string corresponding to the use of the production $A \rightarrow W(B, C)$.

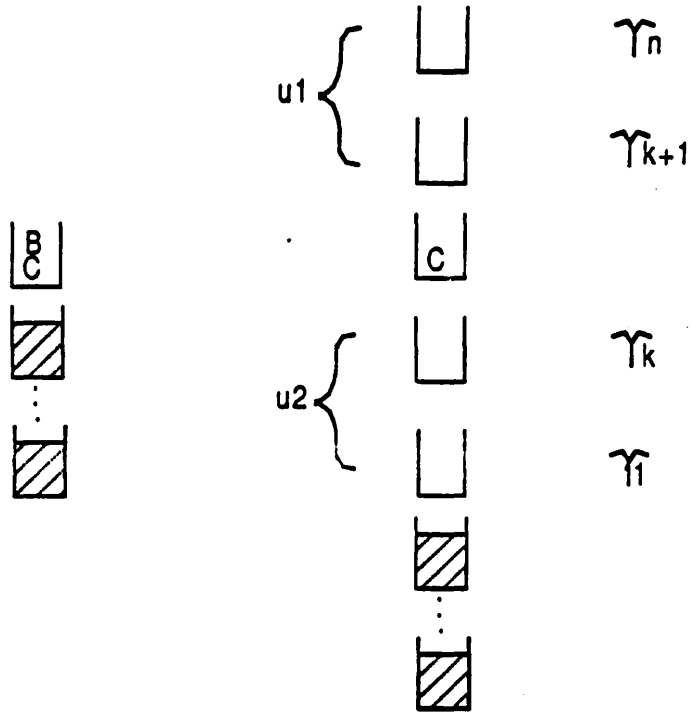

Figure 3.5: Simulating the wrapping operation

3.2.2 Equivalence of epda's and MHG's

In this section we show that the class of MHL's (and hence TAL's) is exactly the class of languages recognized by epda's.

In order to show the inclusion of MHL's in the class of languages recognized by epda's, for every MHG we construct an epda based on the above discussion.

Theorem 3.4 For every MHG, $G = (N, \Sigma, P, S)$, there is an epda, $M = (\{q\}, \Sigma, N \cup \Sigma \cup \{\lambda\}, \delta, q, \{q\}, \cdot)$, such that $L(G) = N(M)$.

We will build M such that it obeys the following proposition. We say that when $A \stackrel{*}{\underset{G}{\Longrightarrow}} w_1 \uparrow w_2$, then the epda with A as its topmost symbol, will insert some new stacks $\Upsilon_1 \ldots \Upsilon_n$ below this top stack when consuming the input w_1 and the stack pointer returns to just below the position where A was initially. These new stacks $\Upsilon_1 \ldots \Upsilon_n$ are erased from the store when the epda consumes the input w_2. Thus, suppose the store of the epda is given by $\Upsilon'_1 \ldots \Upsilon'_m \ddagger \sigma'_1 \ldots \sigma'_j A$ initially, then the epda makes transitions such that

$$\langle q, \Upsilon'_1 \ldots \Upsilon'_m \ddagger \sigma'_1 \ldots \sigma'_j A, \epsilon, w_1 \rangle \models \langle q, \Upsilon'_1 \ldots \Upsilon'_m \Upsilon_1 \ldots \Upsilon_n \ddagger \sigma'_1 \ldots \sigma'_j, w_1 \epsilon \rangle$$

and that $\langle q, \Upsilon'_1 \ldots \Upsilon'_m \Upsilon_1 \ldots \Upsilon_n, \epsilon, w_2 \rangle \models^* \Upsilon'_1 \ldots \Upsilon'_m, w_2 \epsilon$. In order to be precise, we have to insist that when w_1 is consumed that the stacks $\Upsilon_1 \ldots \Upsilon_n$ used above are those which were added below the stack that was on top when the computation started, meaning the stack given by $\ddagger \sigma'_1 \ldots \sigma'_j$ is the same stack as before and not a stack with the same elements. Thus, we use a new stack symbol, say $\ddagger$, for which the transition function is not defined both for reading from or writing on the store. Thus the precise statement is given below.

Proposition 3.1 $A \stackrel{*}{\underset{G}{\Longrightarrow}} w_1 \uparrow w_2$ if and only if $\langle q, \ddagger \ddagger A, \epsilon, w_1 \rangle \models^*_M \langle q, \Upsilon_1 \ldots \Upsilon_n \ddagger \ddagger, w_1, \epsilon \rangle$ for some stacks $\Upsilon_1 \ldots \Upsilon_n$ and $\langle q, \Upsilon_1 \ldots \Upsilon_n, \epsilon, w_2 \rangle \models^*_M \langle q, \epsilon, w_2, \epsilon \rangle$.

The moves of the epda are given as follows.

- If $A \rightarrow a \uparrow$ then M has the following move.

$$(q, \epsilon, \epsilon, \ddagger a) \in \delta(q, \epsilon, A)$$

- If $A \rightarrow \uparrow a$ then M has the following move.

$$(q, \ddagger a, \epsilon, \epsilon) \in \delta(q, \epsilon, A)$$

- If $A \rightarrow \epsilon \uparrow \epsilon$ then M has the following move.

$$(q, \epsilon, \epsilon, \epsilon) \in \delta(q, \epsilon, A)$$

- If $A \rightarrow C1(B,C)$ then M has the following move.

$$(q, \ddagger C, B, \epsilon) \in \delta(q, \epsilon, A)$$

- If $A \rightarrow C2(B,C)$ then M has the following move.

$$(q, \epsilon, C, \ddagger B) \in \delta(q, \epsilon, A)$$

- If $A \rightarrow W(B,C)$ then M has the following move.

$$(q, \epsilon, CB, \epsilon) \in \delta(q, \epsilon, A)$$

- In addition, M has one move. For every $a \in \Sigma$, $(q, \epsilon, \epsilon, \epsilon) \in \delta(q, a, a)$.

Lemma 3.1 $L(G) \subseteq N(M)$

For the inclusion of $L(G)$ in $N(M)$, we induct on the length of the derivation in G. The proposition we prove by induction states that if $A \stackrel{k_1}{\underset{G}{\Longrightarrow}} w_1 \uparrow w_2$ then

$$\langle q, \ddagger\ddagger A, \epsilon, w_1 \rangle \models_M^* \langle q, \Upsilon_1 \ldots \Upsilon_n \ddagger\ddagger, w_1, \epsilon \rangle$$

for some stacks $\Upsilon_1 \ldots \Upsilon_n$ and $\langle q, \Upsilon_1 \ldots \Upsilon_n, \epsilon, w_2 \rangle \models_M^* \langle q, \epsilon, w_2, \epsilon \rangle$.

Base Case: the length of the derivation is one.

Case 1: $A \stackrel{1}{\underset{G}{\Longrightarrow}} \varepsilon \uparrow$ Then, $(q, \epsilon, \epsilon, \ddagger\varepsilon) \in \delta(q, \epsilon, A)$ and $(q, \epsilon, \epsilon, \epsilon) \in \delta(q, \varepsilon, \varepsilon)$. Thus, $\langle q, \ddagger\ddagger A, \epsilon, \varepsilon \rangle \models \langle q, \ddagger\ddagger\ddagger\varepsilon, \epsilon, \varepsilon \rangle \models \langle q, \ddagger\ddagger, \varepsilon, \epsilon \rangle$ as required.

Case 2: $A \stackrel{1}{\underset{G}{\Longrightarrow}} \uparrow\varepsilon$ Then, $(q, \ddagger\varepsilon, \epsilon, \epsilon) \in \delta(q, \epsilon, A)$ and $(q, \epsilon, \epsilon, \epsilon) \in \delta(q, \varepsilon, \varepsilon)$. Thus, $\langle q, \ddagger\ddagger A, \epsilon, \epsilon \rangle \models \langle q, \ddagger\varepsilon\ddagger\ddagger, \epsilon, \epsilon \rangle$ as required, and $\langle q, \ddagger\varepsilon, \epsilon, \varepsilon \rangle \models \langle q, \ddagger\ddagger, \varepsilon, \epsilon \rangle$ as required.

Inductive Step: Assume that for all $k_1 < k$ we have

$$\text{if } A \stackrel{k_1}{\underset{G}{\Longrightarrow}} w_1 \uparrow w_2 \text{ then } \langle q, \ddagger\ddagger A, \epsilon, w_1 \rangle \models_M^* \langle q, \Upsilon_1 \ldots \Upsilon_n \ddagger\ddagger, w_1, \epsilon \rangle$$

for some stacks $\Upsilon_1 \ldots \Upsilon_n$ and $\langle q, \Upsilon_1 \ldots \Upsilon_n, \epsilon, w_2 \rangle \models_M^* \langle q, \epsilon, w_2, \epsilon \rangle$. Then several cases arise.

1. Let $A \overset{k}{\underset{G}{\Longrightarrow}} w_1 \uparrow w_2, A \rightarrow W(B,C), B \overset{k_1}{\underset{G}{\Longrightarrow}} u_1 \uparrow u_2, C \overset{k_2}{\underset{G}{\Longrightarrow}} v_1 \uparrow v_2$. Then, $w_1 = u_1 v_1$, $w_2 = v_2 u_2$, and $k = k_1 + k_2 + 1$. Thus the inductive hypothesis holds for B, C. Therefore, by the inductive hypothesis, $\langle q, \ddagger_\sharp^\sharp A, \epsilon, w_1 \rangle \models_M \langle q, \ddagger_\sharp^\sharp CB, \epsilon, w_1 \rangle \models_M^*$ $\langle q, \Upsilon_{1,1} \ldots \Upsilon_{n_1,1} \ddagger_\sharp^\sharp C, u_1, v_1 \rangle \models_M^* \langle q, \Upsilon_{1,1} \ldots \Upsilon_{n_1,1} \Upsilon_{1,2} \ldots \Upsilon_{n_2,2} \ddagger_\sharp^\sharp, u_1 v_1, \epsilon \rangle$ where $\Upsilon_1 \ldots \Upsilon_n = \Upsilon_{1,1} \ldots \Upsilon_{n_1,1} \Upsilon_{1,2} \ldots \Upsilon_{n_2,2}$ and $\langle q, \Upsilon_{1,1} \ldots \Upsilon_{n_1,1} \Upsilon_{1,2} \ldots \Upsilon_{n_2,2} \epsilon, v_2 u_2 \rangle \models_M^* \langle q, \Upsilon_{1,1} \ldots \Upsilon_{n_1,1}, v_2, u_2 \rangle \models_M^* \langle q, \epsilon, w_2, \epsilon \rangle$ as required.

2. Let $A \overset{k}{\underset{G}{\Longrightarrow}} w_1 \uparrow w_2, A \rightarrow C1(B,C), B \overset{k_1}{\underset{G}{\Longrightarrow}} u_1 \uparrow u_2, C \overset{k_2}{\underset{G}{\Longrightarrow}} v_1 \uparrow v_2$. Then, $w_1 = u_1$, $w_2 = u_2 v_1 v_2$, and $k = k_1 + k_2 + 1$. Thus the inductive hypothesis holds for B, C. Therefore, by the inductive hypothesis, $\langle q, \ddagger_\sharp^\sharp A, \epsilon, w_1 \rangle \models_M \langle q, \ddagger C \ddagger_\sharp^\sharp B, \epsilon, w_1 \rangle$ $\models_M^* \langle q, \ddagger C \Upsilon_{1,1} \ldots \Upsilon_{n_1,1} \ddagger_\sharp^\sharp, u_1, \epsilon \rangle$ and $\Upsilon_1 \ldots \Upsilon_n = \Upsilon_{1,1} \ldots \Upsilon_{n_1,1}$. $\langle q, \ddagger C \Upsilon_{1,1} \ldots \Upsilon_{n_1,1} \epsilon, u_2 v_1 v_2 \rangle \models_M^* \langle q, \ddagger C, u_2, v_1 v_2 \rangle \models_M^* \langle q, \Upsilon_{1,2} \ldots \Upsilon_{n_2,2}, u_2 v_1, v_2 \rangle$ $\models_M^* \langle q, \epsilon, w_2, \epsilon \rangle$ as required.

3. Let $A \overset{k}{\underset{G}{\Longrightarrow}} w_1 \uparrow w_2, A \rightarrow C2(B,C), B \overset{k_1}{\underset{G}{\Longrightarrow}} u_1 \uparrow u_2, C \overset{k_2}{\underset{G}{\Longrightarrow}} v_1 \uparrow v_2$. Then, $w_1 = u_1 u_2 v_1$, $w_2 = v_2$, and $k = k_1 + k_2 + 1$. Thus the inductive hypothesis holds for B, C. Therefore, by the inductive hypothesis, we obtain the following $\langle q, \ddagger_\sharp^\sharp A, \epsilon, w_1 \rangle \models_M \langle q, \ddagger_\sharp^\sharp C \ddagger B, \epsilon, w_1 \rangle \models_M^* \langle q, \ddagger_\sharp^\sharp C \Upsilon_{1,1} \ldots \Upsilon_{n_1,1}, u_1, u_2 v_1 \rangle \models_M^*$ $\langle q, \ddagger_\sharp^\sharp C, u_1 u_2, v_1 \rangle \models_M^* \langle q, \Upsilon_{1,2} \ldots \Upsilon_{n_2,2} \ddagger_\sharp^\sharp, u_1 u_2 v_1, \epsilon \rangle$ where $\Upsilon_1 \ldots \Upsilon_n = \Upsilon_{1,1} \ldots \Upsilon_{n_1,1}$. Also, $\langle q, \Upsilon_{1,2} \ldots \Upsilon_{n_2,2}, \epsilon, v_2 \rangle \models_M^* \langle q, \epsilon, v_2, \epsilon \rangle$ as required.

Thus, the inductive statement holds for k step derivations too. By considering the derivation from S, we thus show that $L(G) \subseteq N(M)$.

We can now show that $N(M) \subseteq L(G)$. The proposition we prove is the following.

$\langle q, \ddagger\sharp A, \epsilon, w_1\rangle \models_M^{k_1} \langle q, \Upsilon_1 \ldots \Upsilon_n \ddagger\sharp, w_1, \epsilon\rangle$ for some stacks $\Upsilon_1 \ldots \Upsilon_n$ and $\langle q, \Upsilon_1 \ldots \Upsilon_n, \epsilon, w_2\rangle \models_M^{k_2} \langle q, \epsilon, w_2, \epsilon\rangle$ implies that $A \underset{G}{\overset{*}{\Longrightarrow}} w_1 \uparrow w_2$.

Induction is on the pair (k_1, k_2). We define the following ordering on these pairs as follows. We say $(i_1, i_2) \preceq (j_1, j_2)$ iff $i_1 \leq j_1$ and $i_2 \leq j_2$.

The base case corresponds to one of the following two computations of M.

1. $\langle q, \ddagger\sharp A, \epsilon, \epsilon\rangle \models \langle q, \ddagger\sharp\ddagger\varepsilon, \epsilon, \varepsilon\rangle \models \langle q, \ddagger\sharp\varepsilon, \epsilon\rangle$ and $\langle q, \epsilon, \epsilon, \epsilon\rangle \models^0 \langle q, \epsilon, \epsilon, \epsilon\rangle$
   In this case, because of these 1 step computations, we have $A \rightarrow \varepsilon\uparrow$ and thus we obtain $A \underset{G}{\overset{*}{\Longrightarrow}} \varepsilon\uparrow$ as required.

2. $\langle q, \ddagger\sharp A, \epsilon, \varepsilon\rangle \models \langle q, \ddagger\varepsilon\ddagger\sharp, \varepsilon, \epsilon\rangle$ and $\langle q, \ddagger\varepsilon, \epsilon, \varepsilon\rangle \models \langle q, \epsilon, \epsilon, \epsilon\rangle$.
   In this case, we have $A \rightarrow \uparrow\varepsilon$ and hence $A \underset{G}{\overset{*}{\Longrightarrow}} \uparrow\varepsilon$ as required.

Now suppose $\langle q, \ddagger\sharp A, \epsilon, w_1\rangle \models_M^{k_1} \langle q, \Upsilon_1 \ldots \Upsilon_n \ddagger\sharp, w_1, \epsilon\rangle$
for some stacks $\Upsilon_1 \ldots \Upsilon_n$ and
$\langle q, \Upsilon_1 \ldots \Upsilon_n, \epsilon, w_2\rangle \models_M^{k_2} \langle q, \epsilon, w_2, \epsilon\rangle$ implies that $A \underset{G}{\overset{*}{\Longrightarrow}} w_1 \uparrow w_2$. Consider the first step of this computation. The different cases are as follows.

1. The first step of the computation uses $(q, \ddagger C, B, \epsilon) \in \delta(q, \epsilon, A)$. $\langle q, \ddagger\sharp A, \epsilon, w_1\rangle \models \langle q, \ddagger C\ddagger\sharp B, \epsilon, w_1\rangle \models^{i_1} \langle q, \ddagger C\Upsilon_{1,1} \ldots \Upsilon_{n_1,1}\ddagger\sharp, w_1, \epsilon\rangle$ and $\langle q, \ddagger C\Upsilon_{1,1} \ldots \Upsilon_{n_1,1}, \epsilon, w_2\rangle \models^{i_2} \langle q, \ddagger C, u_2, v_1v_2\rangle \models^{j_1} \langle q, \Upsilon_{1,2} \ldots \Upsilon_{n_2,2}, u_2v_1, v_2\rangle \models^{j_2} \langle q, \epsilon, u_2v_1v_2, \epsilon\rangle$ where $w_2 = u_2v_1v_2$. Thus, $(i_1, i_2), (j_1, j_2) \prec (k_1, k_2)$ and therefore the inductive hypothesis holds for these computations. Thus, $B \underset{G}{\overset{*}{\Longrightarrow}} u_1 \uparrow u_2$ and $C \underset{G}{\overset{*}{\Longrightarrow}} v_1 \uparrow v_2$. Also $A \rightarrow C1(B, C)$. Thus $A \underset{G}{\overset{*}{\Longrightarrow}} w_1 \uparrow w_2$ as required.

2. The first step of this computation involves $(q, \epsilon, C, \ddagger B) \in \delta(q, \epsilon, A)$. $\langle q, \ddagger\sharp A, \epsilon, w_1\rangle \models \langle q, \ddagger\sharp C\ddagger B, \epsilon, w_1\rangle \models^{i_1} \langle q, \ddagger\sharp C\Upsilon_{1,1} \ldots \Upsilon_{n_1,1}, u_1, u_2v_1\rangle \models^{i_2} \langle q, \ddagger\sharp C, u_1u_2, v_1\rangle \models^{j_1} \langle q, \Upsilon_{1,2} \ldots \Upsilon_{n_2,2}\ddagger\sharp, u_1u_2v_1\rangle$ where $w_1 = u_1u_2v_1$ and $\langle q, \Upsilon_{1,2} \ldots \Upsilon_{n_2,2}, \epsilon, w_2\rangle \models^{j_2} \langle q, \epsilon, v_2, \epsilon\rangle$ where $w_2 = v_2$. Thus, $(i_1, i_2), (j_1, j_2) \prec (k_1, k_2)$ and therefore the

inductive hypothesis holds for these computations. Thus, $B \stackrel{*}{\underset{G}{\Longrightarrow}} u_1 \uparrow u_2$ and $C \stackrel{*}{\underset{G}{\Longrightarrow}} v_1 \uparrow v_2$. Also $A \rightarrow C2(B, C)$. Thus $A \stackrel{*}{\underset{G}{\Longrightarrow}} w_1 \uparrow w_2$ as required.

3. The first step of this computation involves $(q, \epsilon, CB, \epsilon) \in \delta(q, \epsilon, A)$. $\langle q, \ddagger\ddagger A, \epsilon, w_1 \rangle \models \langle q, \ddagger\ddagger CB, \epsilon, w_1 \rangle \models^{i_1} \langle q, \ddagger \Upsilon_{1,1} \ldots \Upsilon_{n_1,1} \ddagger\ddagger C, u_1, v_1 \rangle \models^{j_1}$ $\langle q, \ddagger \Upsilon_{1,1} \ldots \Upsilon_{n_1,1} \Upsilon_{1,2} \ldots \Upsilon_{n_2,2} \ddagger\ddagger, u_1 v_1, \epsilon \rangle$ where $w_1 = u_1 v_1$ and $\langle q, \Upsilon_{1,1} \ldots \Upsilon_{n_1,1}, \Upsilon_{1,2} \ldots \Upsilon_{n_2,2} \epsilon, w_2 \rangle \models^{j_2} \langle q, \Upsilon_{1,1} \ldots \Upsilon_{n_1,1}, v_2, u_2 \rangle \models^{i_2} \langle q, \epsilon, v_2 u_2, \epsilon \rangle$ where $w_2 = v_2 u_2$. Thus, $(i_1, i_2), (j_1, j_2) \prec (k_1, k_2)$ and therefore the inductive hypothesis holds for these computations. Thus, $B \stackrel{*}{\underset{G}{\Longrightarrow}} u_1 \uparrow u_2$ and $C \stackrel{*}{\underset{G}{\Longrightarrow}} v_1 \uparrow v_2$. Also $A \rightarrow W(B, C)$. Thus $A \stackrel{*}{\underset{G}{\Longrightarrow}} w_1 \uparrow w_2$ as required.

Theorem 3.5 Every language accepted by an epda is a TAL.

We prove this result by showing that for every epda there is a MHG which generates the same language that is accepted by the epda. Having shown that the class of languages generated by TAG's and MHG's are the same, we can conclude that every language accepted by an epda is a TAL.

Note that a push move of an epda will insert new stacks above and below the present topmost stack. This can be considered to be equivalent to a wrapping operation, where we wrap the split string corresponding to parts of the input used in erasing the new stacks around the string used in erasing the topmost stack. For example, if Υ were the top stack before the push move, and as result of the push move, stacks Υ_1 and Υ_2 are added above and below this stack. For such a derivation, the epda goes through four crucial configurations with their associated states;

1. a configuration in which we begin processing the new top stack (i.e., Υ_1),
2. a configuration in which we have considered the stacks that were introduced above the topmost stack, and once again the topmost stack before the push move (i.e., Υ) is the top stack,

3. a configuration in which this stack has also been considered and the epda has now Υ_2 as its topmost stack, and

4. finally a configuration in which we have finished considering Υ_2 also.

The nonterminals of the MHG constructed have the form $\langle q_1, q_2, \sigma, q_3, q_4 \rangle$ where the four states corresponding to the four configurations described above and σ is the symbol on top of the stack Υ. The MHG constructed satisfies the following proposition.

Proposition 3.2 $\langle q_1, q_2, \sigma, q_3, q_4 \rangle \stackrel{*}{\Longrightarrow} w_1 \dagger w_2$ if and only if
$(q_1, \sigma_1 \ldots \sigma_m \sigma, \epsilon, w_1 w_2 w_3) \models^* (q_2, \Upsilon_1 \ldots \Upsilon_n \ddagger \sigma_1 \ldots \sigma_m, w_1, w_2 w_3) \models^*$
$(q_3, \Upsilon_1 \ldots \Upsilon_n, w_1 w_2, w_3) \models^* (q_4, \epsilon, w_1 w_2 w_3, \epsilon)$

Any move of an epda can be written in the following form.

$$(p, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}, \sigma^m \ldots \sigma^1, \ddagger\sigma_i \ldots \ddagger\sigma_1) \in \delta(q, \varepsilon, \sigma)$$

where $\varepsilon \in \Sigma \cup \{\epsilon\}, \sigma \in \Gamma \cup \{\epsilon\}, 0 \leq m \leq 2$. The MHG, G, (corresponding to this epda) will have the following set of productions to simulate this rule. If this move applies then consider the sequence of computations of the epda involved in the erasing of σ (and all the symbols introduced and deleted in the process). We use the following symbols in the MHG. The symbol $\overline{\sigma}$ is used to correspond to the computation with the new store after seeing the symbol ε. Thus the states associated with $\overline{\sigma}$ are p, q_2, q_3, q_4. We use the symbol σ_{above} (respectively σ_{below}) corresponding to the computation of the epda that erases the new stacks introduced above (respectively below) the top stack. Thus the states associated with this symbol is p, q' (respectively q'', q_4). The derivations from σ_{above} and σ_{below} are combined (given by the symbol $\hat{\sigma}$) and wrapped around the part introduced on the top stack. The derivation resulting in the erasure of the symbols $\sigma^m \ldots \sigma^1$ is from the nonterminal using the symbol $\underline{\sigma}$ which naturally will have the states q', q_2, q_3, q'' associated with it. Thus, the MHG will have the following set of productions.

- $\langle q_1, q_2, \sigma, q_3, q_4\rangle \rightarrow W(\varepsilon_{\uparrow}, \langle p, q_2, \overline{\sigma}, q_3, q_4\rangle)$ (1)

  where $q = q_1$.

- $\langle p, q_2, \overline{\sigma}, q_3, q_4\rangle \rightarrow W(\langle p, q', \hat{\sigma}, q'', q_4\rangle, \langle q', q_2, \underline{\sigma}, q_3, q''\rangle)$ (2)

- $\langle p, q', \hat{\sigma}, q'', q_4\rangle \rightarrow C2(\langle p, q', \sigma_{above}\rangle, \epsilon_{\uparrow}\epsilon, \langle q'', q_4, \sigma_{below}\rangle)$ (3)

- if $i > 0$ then $\langle p, q', \sigma_{above}\rangle \rightarrow Ci(\langle q_1^1, q_2^1, \sigma_1, q_3^1, q_4^1\rangle, \ldots, \langle q_1^i, q_2^i, \sigma_i, q_3^i, q_4^i\rangle)$ (4a)

  where $p = q_1^1, 1 \leq k \leq i-1, q_4^k = q_1^{k+1}, q_4^i = q'$

  if $i = 0$ then $\langle p, q', \sigma_{above}\rangle \rightarrow \epsilon_{\uparrow}\epsilon$ (4b)

  with $p = q'$.

  if $n > i$ $\langle q'', q_4, \sigma_{below}\rangle \rightarrow C1(\langle q_1^{i+1}, q_2^{i+1}, \sigma_{i+1}, q_3^{i+1}, q_4^{i+1}\rangle, \ldots, \langle q_1^i, q_2^n, \sigma_n, q_3^n, q_4^n\rangle)$ (5a)

where $q_4 = q_4^n, q'' = q_1^{i+1}$ and $i+1 \leq k \leq n$, $q_4^k = q_1^{k+1}$

if $n = i$ then $\langle q'', q_4, \sigma_{below}\rangle \rightarrow \epsilon_{\uparrow}\epsilon$ (5b)

where $q'' = q_4$

- if $m = 2$ then $\langle q', q_2, \mathcal{Q}, q_3, q''\rangle \rightarrow W(\langle q_{1,1}, q_{2,1}, \sigma_1, q_{3,1}, q_{4,1}\rangle, \langle q_{1,2}, q_{2,2}, \sigma_2, q_{3,2}, q_{4,2}\rangle)$ (6a)

Then, $q' = q_{1,1}, q_{2,2} = q_2, q_{3,2} = q_3, q_{4,1} = q'', q_{2,1} = q_{1,2}, q_{4,2} = q_{3,1}$.

If m=1 then $\langle q', q_2, \mathcal{Q}, q_3, q''\rangle \rightarrow \langle q_{1,1}, q_{2,1}, \sigma_1, q_{3,1}, q_{4,1}\rangle$ (6b)

where $q' = q_{1,1}, q_{4,1} = q'', q_2 = q_{2,1}, q_3 = q_{3,1}$.

If m=0 then $\langle q', q_2, \mathcal{Q}, q_3, q''\rangle \rightarrow \epsilon$ (6c)

only if $q' = q_2, q_3 = q''$.

We prove that $L(G) \subseteq N(M)$ by showing that for all k, if $\langle q_1, q_2, \sigma, q_3, q_4\rangle \overset{k}{\underset{\sigma}{\Longrightarrow}} w_{1\uparrow}w_2$ then

$$(q_1, \ddagger\sharp\sigma, \epsilon, w_1) \models^* (q_2, \Upsilon_1 \ldots \Upsilon_n \ddagger\sharp, w_1, \epsilon) \text{ and}$$
$$(q_3, \Upsilon_1 \ldots \Upsilon_n, \epsilon, w_2) \models^* (q_4, \epsilon, w_2, \epsilon)$$

The base case corresponds to $\langle q_1, q_2, \sigma, q_3, q_4\rangle \Longrightarrow \varepsilon_{\uparrow}$. Therefore, the following set of productions would have been used.

$\langle q_1, q_2, \sigma, q_3 q_4\rangle \rightarrow W(\varepsilon_{\uparrow}, \langle p, q_2, \overline{\sigma}, q_3, q_4\rangle)$

$\langle p, q_2, \overline{\sigma}, q_3, q_4\rangle \rightarrow W(\langle p, q', \hat{\sigma}, q'', q_4\rangle, \langle q', q_2, \mathcal{Q}, q_3, q''\rangle)$

$\langle p, q', \hat{\sigma}, q'', q_4\rangle \rightarrow C2(\langle p, q', \sigma_{above}\rangle, \epsilon_{\uparrow}\epsilon, \langle q'', q_4, \sigma_{below}\rangle)$.

$\langle p, q', \sigma_{above}\rangle \rightarrow \epsilon_{\uparrow}\epsilon$ where $p = q'$

$\langle q'', q_4, \sigma_{below}\rangle \rightarrow \epsilon_{\uparrow}\epsilon$ where $q'' = q_4$

$\langle q', q_2, \mathcal{Q}, q_3, q''\rangle \rightarrow \epsilon$.

Thus, we have $p = q_2 = q', q_3 = q'' = q_4$. Therefore the epda has the move $(p, \epsilon, \epsilon, \epsilon) \in \delta(q_1, \varepsilon, \sigma)$. Thus, $(q_1, \ddagger\sharp\sigma, \epsilon, \varepsilon) \models (p, \ddagger\sharp, \varepsilon, \epsilon)$ and $(q_3, \epsilon, \epsilon, \epsilon) \models^0 (q_3, \epsilon, \epsilon, \epsilon)$ as required.

For the inductive step, let

$$\langle q_1, q_2, \sigma, q_3, q_4 \rangle \stackrel{k}{\Longrightarrow} w_{1\uparrow} w_2$$

The first production used in this derivation has to be

$$\langle q_1, q_2, \sigma, q_3, q_4 \rangle \rightarrow W(\varepsilon_{\uparrow}, \langle p, q_2, \overline{\sigma}, q_3, q_4 \rangle)$$

for some p. This implies that the following $k-1$ step derivation took place.

$$\langle p, q_2, \overline{\sigma}, q_3, q_4 \rangle \stackrel{k-1}{\Longrightarrow} w_{\uparrow} w_2$$

where $w_1 = \varepsilon w$. Since the first step in this derivation must have used the production given in (2),

$$\langle p, q_2, \overline{\sigma}, q_3, q_4 \rangle \rightarrow W(\langle p, q', \hat{\sigma}, q'', q_4 \rangle, \langle q', q_2, \mathcal{L}, q_3, q'' \rangle)$$

Thus we will have the derivations $\langle p, q', \hat{\sigma}, q'', q_4 \rangle \stackrel{k_1}{\Longrightarrow} u_{1\uparrow} u_2$ and $\langle q', q_2, \mathcal{L}, q_3, q'' \rangle \stackrel{k_2}{\Longrightarrow} v_{1\uparrow} v_2$ where $u_1 v_1 = w, v_2 u_2 = w_2$, and $k_1 + k_2 + 1 = k - 1$. In the derivation from $\langle p, q', \hat{\sigma}, q'', q_4 \rangle$, we would have used the production,

$$\langle p, q', \overline{\sigma}, q'', q_4 \rangle \rightarrow C2(\langle p, q', \sigma_{above} \rangle, \langle q'', q_4, \sigma_{below} \rangle)$$

first. Thus we can assume the following derivations took place. $\langle p, q', \sigma_{above} \rangle \stackrel{k_3}{\Longrightarrow} \overline{u_1}$, $\langle q'', q_4, \sigma_{below} \rangle \stackrel{k_4}{\Longrightarrow} \overline{u_2}$

where $k_3 + k_4 + 1 = k_1$. For the derivation, $\langle p, q', \sigma_{above} \rangle \stackrel{k_3}{\Longrightarrow} \overline{u_1}$, we would have used either production $4a$ or $4b$ in which case we would obtain the derivations, d_1 or d_2 respectively, given below.

Derivation d_1: The first production used is that given in $4a$. Note in this case $p = q_1^1, q_4^i = q', 1 \leq r \leq i-1, q_2^r = q_3^r, q_4^r = q_1^{r+1}$. Then for $1 \leq r \leq i$ it must be the case that $\langle q_1^r, q_2^r, \sigma_1, q_3^r, q_4^r \rangle \stackrel{l_r}{\Longrightarrow} u_{1,r\uparrow} u_{2,r}$ where $u_{1,1} u_{2,1} u_{1,2} u_{2,2} \ldots u_{1,i} u_{2,i} = u_1$ and $l_1 + \cdots + l_i = k_3 - 1$.

Derivation d_2 The first (and only) production used was $\langle p, q', \sigma_{above} \rangle \rightarrow \epsilon$ where $p = q'$.

Similarly for the derivation, $\langle q'', q_4, \sigma_{below}\rangle \stackrel{k_4}{\Longrightarrow} \overline{u_2}$, we would have used either production $5a$ or $5b$ in which case we would obtain the derivations, d_3 or d_4 respectively, given below.

Derivation d_3: The first production used is that given in $5a$ Note in this case $q'' = q_1^{i+1}, q_4^n = q_4, i+1 \leq r \leq n-1, q_2^r = q_3^r, q_4^r = q_1^{r+1}$. Then for $i+1 \leq r \leq n$ it must be the case that $\langle q_1^r, q_2^r, \sigma_r, q_3^r, q_4^r\rangle \stackrel{l_r}{\Longrightarrow} u_{1,r}\uparrow u_{2,r}$ where $u_{1,i+1}u_{2,i+1}\cdots u_{1,n}u_{2,n} = u_2$ and $l_{i+1} + \cdots + l_n = k_4 - 1$.

Derivation d_4 The first (and only) production used was $\langle q'', q_4, \sigma_{below}\rangle \rightarrow \epsilon$ where $q'' = q_4$. Finally, $\langle q', q_2, \mathcal{Q}, q_3, q''\rangle \stackrel{k_2}{\Longrightarrow} v_1\uparrow v_2$. Then three more cases arise, depending on whether $m = 0, m = 1, m = 2$.

Derivation d_5 $m = 2$. In this case the first production applied is that given in $6a$, where $q' = q_{1,1}, q_{2,2} = q_2, q_{3,2} = q_3, q_{4,1} = q'', q_{2,1} = q_{1,2}, q_{4,2} = q_{3,1}$. It must be the case that $\langle q_{1,1}, q_{2,1}, \sigma_1, q_{3,1}, q_{4,1}\rangle \stackrel{k_5}{\Longrightarrow} x_1\uparrow x_2$ and $\langle q_{1,2}, q_{2,2}, \sigma_2, q_{3,2}, q_{4,2}\rangle \stackrel{k_6}{\Longrightarrow} y_1\uparrow y_2$ where $k_5 + k_6 = k_2 - 1, v_1 = x_1y_1, v_2 = y_2x_2$.

Derivation d_6 m=1. In this case the first production used is that given in $6b$, where $q' = q_{1,1}, q_{2,1} = q_2, q_{3,1} = q_3, q_{4,1} = q''$. It must be the case that $\langle q_{1,1}, q_{2,1}, \sigma_1, q_{3,1}, q_{4,1}\rangle \stackrel{k_2-1}{\Longrightarrow} v_1\uparrow v_2$.

Derivation d_7 m=0. In this case the first (and only) production used is $\langle q', q_2, \mathcal{Q}, q_3, q''\rangle \rightarrow \epsilon$. Thus, $v_1 = v_2 = \epsilon$ and $q' = q_2, q'' = q_3$.

In the k-step derivation starting from $\langle q, q_2, \sigma, q_3, q_4\rangle$, any one of the following sequence of subderivations are possible:

$$(d_1, d_3, d_5), (d_1, d_3, d_6), (d_1, d_3, d_7),$$
$$(d_2, d_3, d_5), (d_2, d_3, d_6), (d_2, d_3, d_7),$$
$$(d_1, d_4, d_5), (d_1, d_4, d_6), (d_1, d_4, d_7),$$
$$(d_2, d_4, d_5), (d_2, d_4, d_6), (d_2, d_4, d_7).$$

Let us now consider the sequence (d_1, d_3, d_5). In order to prove the result by induction, we must show that

- $(q_1, \ddagger_{\#}^{\#}\sigma, \epsilon, w_1) \models^{*} (q_2, \Upsilon_1 \ldots \Upsilon_n \ddagger_{\#}^{\#}, w_1, \epsilon)$ and
- $(q_3, \Upsilon_1 \ldots \Upsilon_n \epsilon, w_2) \models^{*} (q_4, \epsilon, w_2, \epsilon)$.

Because of the productions involved in this derivation, it must be the case that $(p, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}, \sigma^2\sigma^1, \ddagger\sigma_i \ldots \ddagger\sigma_1) \in \delta(q_1, \epsilon, \sigma)$. Thus,

- $(q_1, \ddagger_{\#}^{\#}\sigma, \epsilon, w_1) \models (p, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}, \ddagger_{\#}^{\#}\sigma^2\sigma^1, \ddagger\sigma_i \ldots \ddagger\sigma_1, \varepsilon, w)$, where $w_1 = \varepsilon w$.

Since we are considering the subsequence d_1, we have $i > 0$. Thus, for $1 \leq r \leq i$, we have $l_r = l_{1,r} + l_{2,r} < k$, and therefore the inductive hypothesis holds for the following l_r-step derivations.

$$\langle q_1^r, q_2^r, \sigma_r, q_3^r, q_4^r \rangle \stackrel{l_r}{\Longrightarrow} u_{1,r} \uparrow u_{2,r}$$

Thus, we have the following computation.

- $(p, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger_{\#}^{\#}\sigma^2\sigma^1\ddagger\sigma_i \ldots \ddagger\sigma_1, \varepsilon, w)$

$\models^{*} (q_1^r, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger_{\#}^{\#}\sigma^2\sigma^1\ddagger\sigma_i \ldots \ddagger\sigma_r, \varepsilon u_{1,1}u_{2,1} \ldots u_{1,r-1}u_{2,r-1}, u_{1,r} \ldots u_{1,i}u_{2,i}x_1y_1)$

where $u_{1,1}u_{2,1} \ldots u_{1,i}u_{2,i}x_1y_1 = w$

$\models^{*} (q_2^r, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger_{\#}^{\#}\sigma^2\sigma^1\ddagger\sigma_i \ldots \ddagger\sigma_{r+1}\Upsilon_1^r \ldots \Upsilon_{n_r}^r,$

$\varepsilon u_{1,1}u_{2,1} \ldots u_{1,r-1}u_{2,r-1}u_{1,r}, u_{2,r} \ldots u_{1,i}u_{2,i}x_1y_1)$

$\models^{0} (q_3^r, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger_{\#}^{\#}\sigma^2\sigma^1\ddagger\sigma_i \ldots \ddagger\sigma_{r+1}\Upsilon_1^r \ldots \Upsilon_{n_r}^r,$

$\varepsilon u_{1,1}u_{2,1} \ldots u_{1,r-1}u_{2,r-1}u_{1,r}, u_{2,r} \ldots u_{1,i}u_{2,i}x_1y_1)$ since $q_2^r = q_3^r$

$\models^{*} (q_4^r, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger_{\#}^{\#}\sigma^2\sigma^1\ddagger\sigma_i \ldots \ddagger\sigma_{r+1},$

$\varepsilon u_{1,1}u_{2,1} \ldots u_{1,r-1}u_{2,r-1}u_{1,r}u_{2,r}, u_{1,r+1}u_{2,r+1} \ldots u_{1,i}u_{2,i}x_1y_1)$

$\models^{0} (q_1^{r+1}, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger_{\#}^{\#}\sigma^2\sigma^1\ddagger\sigma_i \ldots \ddagger\sigma_{r+1},$

$\varepsilon u_{1,1}u_{2,1} \ldots u_{1,r-1}u_{2,r-1}u_{1,r}u_{2,r}, u_{1,r+1}u_{2,r+1} \ldots u_{1,i}u_{2,i}x_1y_1)$ since $q_4^r = q_1^{r+1}$

$\models^{*} (q_4^i, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger_{\#}^{\#}\sigma^2\sigma^1, \varepsilon u_{1,1}u_{2,1} \ldots u_{1,i}u_{2,i}, x_1y_1)$

$\models^{0} (q', \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger_{\#}^{\#}\sigma^2\sigma^1, \varepsilon\overline{u_1}, x_1y_1)$ since $u_{1,1}u_{2,1} \ldots u_{1,i}u_{2,i} = \overline{u_1}, q' = q_4^i$

Now considering the subsequence d_5, we have $m = 2$. The inductive hypothesis holds for the following two derivations in this k_5-step derivations.

$\Longrightarrow x_1; x_2$

$\langle q_{1,2}, q_{2,2}, \sigma^2, q_{3,2}, q_{4,2}\rangle \overset{*}{\Longrightarrow} y_1; y_2$

We have already shown $(q_1, \ddagger\sigma, \epsilon, w_1) \models^* (q', \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\sigma^2\sigma^1, \varepsilon\overline{u_1}, x_1 y_1)$

$\models^0 (q_{1,1}, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\sigma^2\sigma^1, \varepsilon\overline{u_1}, x_1 y_1)$ since $q_{1,1} = q'$.

$\models^* (q_{2,1}, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon_1^1 \ldots \Upsilon_1^{n_1}\ddagger\sigma^2, \varepsilon\overline{u_1}x_1, y_1)$ by inductive hypothesis,

$\models^* (q_{1,2}, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon_1^1 \ldots \Upsilon_1^{n_1}\ddagger\sigma^2, \varepsilon\overline{u_1}x_1, y_1)$ since $q_{1,2} = q_{2,1}$

$\models^* (q_{2,1}, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon_1^1 \ldots \Upsilon_1^{n_1}\Upsilon_2^1 \ldots \Upsilon_2^{n_2}\ddagger, \varepsilon\overline{u_1}x_1 y_1, \epsilon)$

$\models^0 (q_2, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon_1^1 \ldots \Upsilon_1^{n_1}\Upsilon_2^1 \ldots \Upsilon_2^{n_2}\ddagger, \varepsilon\overline{u_1}x_1 y_1, \epsilon)$ since $q_2 = q_{2,2}$

Since $w_1 = \varepsilon\overline{u_1}x_1 y_1$, we have proved one half of what is required too be shown considering the computation from $(q_3, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon_1^1 \ldots \Upsilon_1^{n_1}\Upsilon_2^1 \ldots \Upsilon_2^{n_2}, \epsilon, w_2)$, since have $q_3 = q_{3,2}$ and by the second part of the k_5-step derivation

$(q_3, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon_1^1 \ldots \Upsilon_1^{n_1}\Upsilon_2^1 \ldots \Upsilon_2^{n_2}, \epsilon, y_2 x_2 \overline{u_2})$

$\models^* (q_{4,2}, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon_1^1 \ldots \Upsilon_1^{n_1}, y_2, x_2\overline{u_2})$ where we write w_2 as $y_2 x_2 \overline{u_2}$

$\models^0 (q_{3,1}, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon_1^1 \ldots \Upsilon_1^{n_1}, y_2, x_2\overline{u_2})$ since $q_{4,2} = q_{3,1}$

$\models^* (q_{4,1}, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}, y_2 x_2, \overline{u_2})$ and since $q_{4,1} = q''$ we have

$\models^0 (q'', \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}, y_2 x_2, \overline{u_2})$

Now considering the derivation in d_3, we have $n > i$. Identical to the derivation sequence d_1, we write $\overline{u_2} = u_{1,i+1}u_{2,i+1} \ldots u_{1,r}u_{2,r} \ldots u_{1,n}u_{2,n}$. Since $w_2 = y_2 x_2 \overline{u_2}$ and since the inductive hypothesis holds for the derivation of $u_{1,r}; u_{2,r}$ for $i - 1 \leq r \leq n$, we have $(q_3, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}, \epsilon, w_2) \models^* (q_4^n, \epsilon, w_2\ \epsilon)$. Since $q_4^n = q_4$, we have shown all required parts for showing that the inductive hypothesis holds of the k-step derivation

$$\langle q_1, q_2, \sigma, q_3, q_4\rangle \overset{k}{\Longrightarrow} w_1; w_2$$

considering the subsequences (d_1, d_3, d_5). The other cases can be shown similarly.

Thus we have shown by induction that

if $\langle q_1, q_2, \sigma, q_3, q_4\rangle \stackrel{k}{\Longrightarrow} w_1 \uparrow w_2$ then $(q_1, \ddagger\sigma, \epsilon, w_1) \models^* (q_2, \Upsilon_1 \ldots \Upsilon_n \ddagger, w_1, \epsilon)$ and

$$(q_3, \Upsilon_1 \ldots \Upsilon_n, \epsilon, w_2) \models^* (q_4, \epsilon, w_2, \epsilon).$$

Considering the derivation from $\langle q_0, p, \sigma_0, p, q\rangle$ for some p, q, we can establish $L(G) \subseteq N(M)$.

Lemma 3.2 $N(M) \subseteq L(G)$

In order to prove this lemma, we show by induction that for all pairs (k_1, k_2) the proposition given below holds.

Proposition 3.3 If $(q_1, \ddagger\sigma, \epsilon, w_1) \models^{k_1} (q_2, \Upsilon_1 \ldots \Upsilon_n \ddagger, w_1, \epsilon)$ and
$(q_3, \Upsilon_1 \ldots \Upsilon_n, \epsilon, w_2) \models^{k_2} (q_4, \epsilon, w_2, \epsilon)$ then
$\langle q_1, q_2, \sigma, q_3, q_4\rangle \stackrel{*}{\Longrightarrow} w_1 \uparrow w_2$ where $\sigma \in \Gamma \cup \{\epsilon\}$.

In the most general case, the first move in this computation sequence is due to

$(p, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}, \sigma^m \ldots \sigma^1, \ddagger\sigma_i \ldots \ddagger\sigma_1) \in \delta(q_1, \varepsilon, \sigma)$

where $\varepsilon \in \Sigma \cup \{\epsilon\}$, $\varepsilon w = w_1$ and $0 \leq m \leq 2$.

Thus, we have $(q_1, \ddagger\sigma, \epsilon, w_1) \models (p, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\sigma^m \ldots \sigma^1 \ddagger\sigma_i \ldots \ddagger\sigma_1, \varepsilon, w)$
$\models^{k_1-1} (q_2, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1} \Upsilon^1_m \ldots \Upsilon^{n_m}_m \ldots \Upsilon^1_1 \ldots \Upsilon^{n_1}_1 \ddagger, w_1, \epsilon)$. The $k_1 - 1$-step computation can be divided into the following subcomputations.

- $(p, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\sigma^m \ldots \sigma^1 \ddagger\sigma_i \ldots \ddagger\sigma_1, \varepsilon, w)$
$\models^{k_3} (q - 4^i, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\sigma^m \ldots \sigma^1, \varepsilon\overline{u_1}, x_{1,1} \ldots x_{m,1})$ where $w_1 = \varepsilon\overline{u_1}x_{1,1} \ldots x_{m,1}$
$\models^0 (q', \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\sigma^m \ldots \sigma^1, \varepsilon\overline{u_1}, x_{1,1} \ldots x_{m,1})$ where $q' = q^i_4$
$\models^{k_4} (q_2, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1} \Upsilon^1_m \ldots \Upsilon^{n_m}_m \ldots \Upsilon^1_1 \ldots \Upsilon^{n_1}_1 \ddagger, w_1, \epsilon)$,
where $\ddagger\sigma_n \ldots \ddagger\sigma_{i+1} \Upsilon^1_m \ldots \Upsilon^{n_m}_m \ldots \Upsilon^1_1 \ldots \Upsilon^{n_1}_1 = \Upsilon_1 \ldots \Upsilon_n$.

Similarly, the k_2-step computation is broken up into the following subsequences

- $(q_3, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1} \Upsilon^1_m \ldots \Upsilon^{n_m}_m \ldots \Upsilon^1_1 \ldots \Upsilon^{n_1}_1, \epsilon, w_2)$

$\models^{k_5} (q'', \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}, x_{1,2} \ldots x_{m,2}, \overline{u_2})$

$\models^{k_6} (q_4, \epsilon, w_2, \epsilon)$ where w_2 is written as $x_{1,2} \ldots x_{m,2}, \overline{u_2}$).

We will consider the $k_1 - 1$-step computation first. Of this subcomputation, we will consider the subcomputation with k_3-steps. We say, $p = q_1^1$ and $k_3 = l_{1,1} + l_{1,2} + \cdots + l_{i,1} + l_{i,2}$. We have two cases corresponding to $i > 0$ and $i = 0$.

Computation C_1, $i > 0$

We say, $p = q_1^1$, $u_{1,1}u_{2,1} \ldots u_{1,i}u_{2,i} = \overline{u_1}$ and $k_3 = l_{1,1} + l_{1,2} + \cdots + l_{i,1} + l_{i,2}$. Then,

$(p, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\sigma^m \ldots \sigma^1\ddagger\sigma_i \ldots \ddagger\sigma_1, \varepsilon, w) \models^{l_1} (q_1^r, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\sigma^m \ldots \sigma^1\ddagger\sigma_i \ldots \ddagger\sigma_r,$

$\varepsilon u_{1,1}u_{2,1} \ldots u_{1,r-1}u_{2,r-1}, u_{1,r}u_{2,r} \ldots u_{1,i}u_{2,i}x_{1,1} \ldots x_{m,1})$

where $l' = l_{1,1} + l_{1,2} + \cdots + l_{r-1,1} + l_{r-1,2}$.In $l_{r,1}$ computation steps we will obtain the ID $(q_2^r, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\sigma^m \ldots \sigma^1\ddagger\sigma_i \ldots \ddagger\sigma_{r+1}\Upsilon_1^1 \ldots \Upsilon_{n_r}^r,$

$\varepsilon u_{1,1}u_{2,1} \ldots u_{1,r-1}u_{2,r-1}u_{1,r}, u_{2,r} \ldots u_{2,i})$. Since $q_2^r = q_3^r$,

this is the same as the ID

$(q_3^r, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\sigma^m \ldots \sigma^1\ddagger\sigma_i \ldots \ddagger\sigma_{r+1}\Upsilon_1^1 \ldots \Upsilon_{n_r}^r,$

$\varepsilon u_{1,1}u_{2,1} \ldots u_{1,r-1}u_{2,r-1}u_{1,r}, u_{2,r} \ldots u_{2,i}) \models^{l_{r_2}}$

$(q_4^r, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\sigma^m \ldots \sigma^1\ddagger\sigma_i \ldots \ddagger\sigma_{r+1}, \varepsilon u_{1,1}u_{2,1} \ldots u_{1,r-1}u_{2,r-1}u_{1,r}u_{2,r}, u_{1,r+1} \ldots u_{2,i})$

$\models^{l_2} (q_4^i, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\sigma^m \ldots \sigma^1, \varepsilon u_{1,1}u_{2,1} \ldots u_{1,i}u_{2,i}, \epsilon)$

where $l_2 = l_{r+1,1} + l_{r+1,2} + \cdots l_{i,1} + l_{i,2}$. Since we write $q_4^i = q'$ and $\overline{u_1} = u_{1,1}u_{2,1} \ldots u_{1,i}u_{2,i}$ we have

$(q', \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\sigma^m \ldots \sigma^1, \varepsilon\overline{u_1}, \epsilon)$

If we assume inductively that for each of the $l_{r,1} + l_{r,2}$ computations the proposition holds, then we know that for $1 \leq r \leq i$ the following holds

$$\langle q_1^r, q_2^r, \sigma_r, q_3^r, q_4^r \rangle \stackrel{*}{\Longrightarrow} u_1 \restriction u_2$$

Then using the production $\langle p, q', \sigma_{above} \rangle \rightarrow C1(\langle q_1^1, q_2^1, \sigma_1, q_3^1, q_4^1 \rangle, \ldots, \langle q_1^i, q_2^i, \sigma_i, q_3^i, q_4^i \rangle)$,

we have $\langle p, q', \sigma_{above} \rangle \stackrel{*}{\Longrightarrow} \overline{u_1}$.

Computation C_2 $i = 0$

Thus with, $p = q'$ it is obviously the case that $(p, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\ddagger\sigma^m \ldots \sigma^1, \varepsilon, w)$ $\models^0 (q', \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\ddagger\sigma^m \ldots \sigma^1, \varepsilon, w)$ where $u_1 = \epsilon, k_3 = 0$. Correspondingly, since $i = 0$, we have the production $\langle p, q', \sigma_{above} \rangle \rightarrow \overline{\epsilon}$

• erasing of $\sigma^m \ldots \sigma^1$. We have three cases to consider, $m = 2, m = 1$, or $m = 0$.

Computation C_3 $m = 2$

By writing $q_{1,1}$ as q' we have, $(q', \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\ddagger\sigma^2\sigma^1, \varepsilon u_1, x_1x_2)$

$\models^{k_7} (q_{2,1}, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon^1_1 \ldots \Upsilon^1_{n_1}\ddagger\ddagger\sigma^2, \varepsilon u_1x_1, x_2)$

$\models^0 (q_{1,2}, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon^1_1 \ldots \Upsilon^1_{n_1}\ddagger\ddagger\sigma^2, \varepsilon u_1x_1, x_2)$ with $q_{2,1} = q_{1,2}$

$\models^{k_8} (q_{2,2}, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon^1_1 \ldots \Upsilon^1_{n_1}\Upsilon^2_1 \ldots \Upsilon^2_{n_2}\ddagger\ddagger, \varepsilon u_1x_1x_2, \epsilon)$

$\models^0 (q_2, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon^1_1 \ldots \Upsilon^1_{n_1}\Upsilon^2_1 \ldots \Upsilon^2_{n_2}\ddagger\ddagger, \varepsilon u_1x_1x_2, \epsilon)$

with $q_2 = q_{2,2}, w_1 = \varepsilon u_1x_1x_2$ and $k_7 + k_8 = k_4$.

Computation C_4 $m = 1$

With $q' = q_{1,1}, q_{2,1} = q_2$, we can write this subcomputation as

$(q', \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\ddagger\ddagger\sigma^1, \varepsilon u_1, x_1)$

$\models^{k_4} (q_2, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon^1_1 \ldots \Upsilon^1_{n_1}\ddagger\ddagger, \varepsilon u_1x_1, \epsilon)$

Computation C_5 $m = 0$

In this case $q' = q_2$, and $k_4 = 0$.

• We now consider the rest of the computation, i.e., the final k_2-steps of the computation. First we consider the removal of stacks introduced above σ_{i+1}. As before there are three cases to consider corresponding to the three possible values for m.

Computation C_6 $m = 2$.

We let $q_3 = q_{3,2}, q'' = q_{4,1}$. Then, $(q_3, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon_1^1 \ldots \Upsilon_{n_1}^1\Upsilon_1^2 \ldots \Upsilon_{n_2}^2, \epsilon, y_2y_1u_2)$
$\models^{k_9} (q_{4,2}, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon_1^1 \ldots \Upsilon_{n_1}^1, y_2, y_1u_2)$
$\models^0 (q_{3,1}, \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}\Upsilon_1^1 \ldots \Upsilon_{n_1}^1, y_2, y_1u_2)$ letting $q_{4,2} = q_{3,1}$
$\models^{k_{10}} (q'', \ddagger\sigma_n \ldots \ddagger\sigma_{i+1}, y_2, y_1u_2)$ letting $q'' = q_{4,1}$

Computation C_6 will be used if and only if computation sequence C_3 is used earlier. If these two sequences are used, then $m = 2$ and therefore the MHG will have the production

$$\langle q', q_2, \underline{\sigma}, q_3, q'' \rangle \rightarrow W(\langle q_{1,1}, q_{2,1}, \sigma^1, q_{3,1}, q_{4,1} \rangle, \langle q_{1,2}, q_{2,2}, \sigma^2, q_{3,2}, q_{4,2} \rangle)$$

with $q' = q_{1,1}, q_{2,1} = q_{1,2}, q_{2,2} = q_2, q_3 = q_{3,2}, q_{4,2} = q_{3,1}, q_{4,1} = q''$. If we assume that the length of the computations resulting from σ^2, σ^1 satisfy the inductive clause, then we have

$$\langle q_{1,1}, q_{2,1}, \sigma^1, q_{3,1}, q_{4,1} \rangle \stackrel{*}{\Longrightarrow} x_1 \uparrow y_1$$
$$\langle q_{1,2}, q_{2,2}, \sigma^2, q_{3,2}, q_{4,2} \rangle \stackrel{*}{\Longrightarrow} x_2 \uparrow y_2$$

Thus we have

$$\langle q', q_2, \underline{\sigma}, q_3, q'' \rangle \stackrel{*}{\Longrightarrow} x_1 y_1 \uparrow y_2 x_2$$

Computation C_7 $m = 1$

In this case, we have $\Upsilon_1^1 \ldots \Upsilon_{n_1}^1$ introduced because of σ^1 on the stack, which is removed by the epda on consuming the input y_1 and changing from state q_3 to q''.

As in the above discussion, C_7 is used if and only if C_4 is used. Using the appropriate production (corresponding to $m = 1$) and assuming the lengths of computations C_4, C_7 satisfy the inductive clause, we have

$$\langle q', q_2, \underline{\sigma}, q_3, q'' \rangle \stackrel{*}{\Longrightarrow} x_1 \uparrow y_1$$

Computation C_8 $m = 0$

In this case, the length of the computation is 0, and with $q_3 = q''$ we have along with the

computation C_5,

$$\langle q', q_2, \sigma, q_3, q'' \rangle \stackrel{*}{\Longrightarrow} \epsilon_1 \epsilon$$

The next sequence of computations that we consider are the erasing of $\sigma_{i+1}, \ldots, \sigma_n$. As in the case of computations C_1 and C_2, there are two cases, computations C_9, C_{10}, that we have to consider. They correspond to whether $n > i$ or not and can we worked out identically to computations C_1 and C_2. As in the those two cases we will then conclude that

$$\langle q'', q_4, \sigma_{below} \rangle \stackrel{*}{\Longrightarrow} \overline{u_2}$$

Thus given the production $\langle p, q', \hat{\sigma}, q'', q_4 \rangle \rightarrow C2(\langle p, q', \sigma_{above} \rangle, \epsilon_1 \epsilon, \langle q'', q_4, \sigma_{below} \rangle)$, we can conclude that

$$\langle p, q', \hat{\sigma}, q'', q_4 \rangle \stackrel{*}{\Longrightarrow} u_1 \overline{u_2}.$$

The different sequences of subcomputations are :

$$(C_1, C_3, C_6, C_9), (C_1, C_4, C_7, C_9), (C_1, C_5, C_8, C_9),$$
$$(C_1, C_3, C_6, C_{10}), (C_1, C_4, C_7, C_{10}), (C_1, C_5, C_8, C_{10}),$$
$$(C_2, C_3, C_6, C_9), (C_2, C_4, C_7, C_9), (C_2, C_5, C_8, C_9),$$
$$(C_2, C_3, C_6, C_{10}), (C_2, C_4, C_7, C_{10}), (C_2, C_5, C_8, C_{10}).$$

The computation sequence (C_2, C_5, C_8, C_{10}) corresponds to the base case. In the other sequences, as we have seen, if the lengths of the various subcomputations satisfy the inductive clause, then we will be guaranteed that by induction we have shown

$$\langle q_1, q_2, \sigma, q_3, q_4 \rangle \stackrel{*}{\Longrightarrow} w_1 \overline{w_2}$$

and have thus proved the lemma. In order to show that $N(M) \subseteq L(M)$, we have to consider the derivation from $\langle q_0, q, \sigma_0, q, p \rangle$ for some q, p.

Thus $N(M) \subseteq L(G)$.

3.2.3 Closure under Intersection with Regular Languages

In this section we show that TAL's are closed under the operation of intersection with a Regular Language. By restricting epda's appropriately, we are also able to get an automaton charaterization of languages generated by linear TAG's.

Theorem 3.6 TAL's are closed under intersection with Regular Languages.

In order to show that TAL's are closed under intersection with Regular Languages, we extend the proof used for CFL's. Thus, let L be a TAL recognized by an epda. $M = (Q, \Sigma, \Gamma, \delta, q_0, Q_F, \sigma_0)$ by final state. Let R be a regular language recognised by a fsa $A = (Q_A, \Sigma, \delta_A, p_0, F_A)$. In order to show that $L \cap R$ is a TAL, we can show there is an epda M_1 such that M_1 recognizes $L \cap R$. Since every string, w, in $L \cap R$ is a string of L and a string of R, w must be recognized by M and also by A. Therefore, we let M_1, defined by $M' = (Q \times Q_A, \Sigma, \Gamma, \delta', [q_0, p_0], Q_F \times F, \sigma_0)$, be a machine which runs M and R in parallel. δ' is defined by

$$\langle [q_1, p_1], \ddagger\sigma'_1\ddagger \ldots \ddagger\sigma'_i, \sigma_1 \ldots \sigma_n, \ddagger\sigma'_{i+1}\ddagger \ldots \ddagger\sigma'_k \rangle \in \delta'([q, p], a, \sigma)$$

if and only if

$$\langle p, \ddagger\sigma'_1\ddagger \ldots \ddagger\sigma'_i, \sigma_1 \ldots \sigma_n, \ddagger\sigma'_{i+1}\ddagger \ldots \ddagger\sigma'_k \ldots \sigma'_k \rangle \in \delta(q, a, \sigma)$$

and $\delta_A(p, a) = p_1$.

Inducting on j, we can show that for all j,

$$\langle [q_0, p_0], \ddagger\sigma_0, \epsilon, w_1 w_2 \rangle \models^j_{M'} \langle [q, p], \Upsilon_1 \ldots \Upsilon_n, w_1, w_2 \rangle$$

if and only if

$$\langle q_0, \ddagger\sigma_0, \epsilon, w \rangle \models^j_M \langle q, \Upsilon_1 \ldots \Upsilon_n, w_1, w_2 \rangle \quad \text{and} \quad \delta_A(p_0, w_1) = p.$$

The basis $j = 0$ is trivial, since $p = p_0, q = q_0, n = 1, \Upsilon_1 = \ddagger\sigma_0, w_1 = \epsilon$. For induction let the statement be true for $i - 1$. Then we have

$$\langle [q_0, p_0], \ddagger\sigma_0, \epsilon, w_1 w_2 \rangle \models_{M'}^{i-1} \langle [q', p'], \Upsilon_1 \ldots \Upsilon_k \Upsilon', x\varepsilon, w_2 \rangle \models_{M'} \langle [q, p], \Upsilon_1 \ldots \Upsilon_n, x\varepsilon, w_2 \rangle$$

where $w_1 = x\varepsilon$. By induction hypothesis,

$$\langle q_0, \ddagger\sigma_0, \epsilon, w_1 w_2 \rangle \models_M^{i-1} \langle [q', p'], \Upsilon_1 \ldots \Upsilon_k \Upsilon', x\varepsilon, w_2 \rangle \quad \text{and} \quad \delta_A(p_0, w_1) = p.$$

By the definition of δ we have

$$\langle q', \Upsilon_1 \ldots \Upsilon_k \Upsilon', x\varepsilon, w_2 \rangle \models_M \langle q, \Upsilon_1 \ldots \Upsilon_n, x\varepsilon, w_2 \rangle$$

and $\delta_A(p', \varepsilon) = p$. Thus the induction hypothesis will hold for $j = i$ also. The converse can be shown similarly.

3.3 The Power of epda's

In order to help understand the relationship between recognizing powers of the epda and nsa (which accepts Indexed Languages), and why the epda is only slightly more powerful than a pda, we show the equivalence between epda's (and thus TAG's) and a linear version of Indexed Grammars.

Although both the epda and the nsa have a store made up of stacks of stacks of stack elements, the moves of a epda are more like those of a pda rather than a nsa. A move of a pda may be characterized as replacing the top stack element by a sequence of stack symbols. The move of a epda may be characterized in a similar way, the top element of the stack is replaced by a sequence of stacks elements first and then in a second step, this *top* stack is replaced by a sequence of stacks. Thus, the epda may be viewed as a second order pda. In fact, this generalization may be carried on further, leading to an $i^{th} - order$ pda for any $i \geq 1$[2], as has been done in [Weir 87]. The nsa on the other hand allows the stack pointer to leave the top of the stack. The nsa can thus be used to copy unbounded amount of information by a sequence of moves. To understand this and to see how the epda is more restricted, we will consider Indexed grammars with which the nsa is equivalent in the following section.

3.3.1 Tree Adjoining Grammars and a Linear Version of Indexed Grammars

In this section, we describe a linearized version of Indexed Grammars which was considered by Gazdar [Gazdar 85a] for its possible linguistic applicability. We call this grammatical formalisms *Linear Indexed Grammars (LIG)* and show that the class of languages generated by LIG's is exactly TAL's.

[2] $i = 0$ would correspond to a finite state automaton

The productions in an Indexed Grammar can be written as (refer [Gazdar 85a], the original definition of Indexed Grammars appears in [Aho 68])

$X[\cdot\cdot]\rightarrow X_1[i,\cdot\cdot]\ldots X_n[i,\cdot\cdot]$ (push) or

$X[i,\cdot\cdot]\rightarrow X_1[\cdot\cdot]\ldots X_n[\cdot\cdot]$ (pop) or

$X[\cdot\cdot]\rightarrow a$

In an Indexed Grammar, a stack of symbols (called the indices) is associated with nonterminals during derivations. $[\cdot\cdot]$ is used to represent the present stack associated with a nonterminal. In the first production the parent (l.h.s of the production) distributes copies of its stack to its children after pushing the symbol i at the top of the stack. Thus, we can see how the stack information is shared by all the symbols in the right-hand side of the production.

In an LIG the productions have the form

$X[\cdot\cdot]\rightarrow X_1[]\ldots X_j[i,\cdot\cdot]\ldots X_n[]$ (push) or

$X[i,\cdot\cdot]\rightarrow X_1[]\ldots X_j[\cdot\cdot]\ldots X_n[]$ (pop) or

$X[\cdot\cdot]\rightarrow a$

LIG differs from IG in that the stack is passed on to only one of the children. [] corresponds to a new but empty stack. The productions of LIG's can be generalized to be of the form

$X[\cdot\cdot]\rightarrow X_1[l_1\ldots l_{m,1}]\ldots X_j[i,\cdot\cdot]\ldots X_n[l_n\ldots l_{m,n}]$ (push) or

$X[i,\cdot\cdot]\rightarrow X_1[l_1\ldots l_{m,1}]\ldots X_j[\cdot\cdot]\ldots X_n[l_n\ldots l_{m,n}]$ (pop) or

$X[\cdot\cdot]\rightarrow a$

Let G be a LIG. Then $\alpha = \alpha_1 A[l_1 \ldots l_n]\alpha_2$ is a sentential form of G [3]. The derives relation is defined as follows.

- If $A[\cdot\cdot] \rightarrow X_1[] \ldots X_j[l, \cdot\cdot] \ldots X_m[]$is a production of G then
  $\alpha \underset{G}{\Longrightarrow} \alpha_1 X_1[] \ldots X_j[l_1 \ldots l_n l] \ldots X_m[]\alpha_2$
- If $A[l_n, \cdot\cdot] \rightarrow X_1[] \ldots X_j[\cdot\cdot] \ldots X_m[]$is a production of G then
  $\alpha \underset{G}{\Longrightarrow} \alpha_1 X_1[] \ldots X_j[l_1 \ldots l_{n-1}] \ldots X_m[]\alpha_2$ provided $n > 0$.
- If $A[\cdot\cdot] \rightarrow \varepsilon$ is a production of G then
  $\alpha \underset{G}{\Longrightarrow} \alpha_1 \varepsilon \alpha_2$

The language generated by G is defined as

$$L(G) = \{w \mid w \in \Sigma^*,\ S[] \overset{*}{\underset{G}{\Longrightarrow}} w\}$$

The linearity in the copying of stacks in LIG is captured by the epda. Note that in an epda no unbounded amount of information is copied. The single copy of the top stack (which is unbounded in size) is passed onto to the new configuration. This differentiates the epda from the automata for indexed grammars. We can observe the similarity of moves of epda's with a *leftmost* derivation in LIG's by noting that the stack associated with a leftmost nonterminal in a sentential form in LIG will be the topmost stack, with the nonterminal information at its top. When a *push* production is used in LIG, then the index used is pushed on the top stack of the epda, new stacks with just the nonterminal information are inserted above and below this stack, depending on whether these nonterminals (with new but empty stacks) are to the left and right of the nonterminal (on the right hand side) inheriting the stack from the nonterminal in the left hand side. For example, let a sentential form be $A_1[\Gamma_1] \ldots A_n[\Gamma_n]$. After the use of the *push* production, let the sentential form be $B_1[\Gamma'_1] \ldots B_m[\Gamma'_m]A_2[\Gamma_2 \ldots A_n[\Gamma_n]$. Then the store

[3] If we write $A[l_1 \ldots l_n]$, we mean that the stack associated with the nonterminal A has l_n at the top and l_1 at the bottom.

configuration in the epda will transform to $\ddagger\Gamma_n A_n \ldots \ddagger\Gamma_2 A_2 \ddagger\Gamma'_m B_m \ldots \ddagger\Gamma'_1 B_1$ from the store configuration $\ddagger\Gamma_n A_n \ldots \ddagger\Gamma_1 A_1$. To detect the bottom of every stack in an epda is the "$\ddagger$" symbol.

3.3.2 Inclusion of LIL's in TAL's

To show the inclusion of the class of languages (LIL's) generated by LIG's in the class recognized by epda's, we show that for each LIG, G, there is a epda, M, such that following proposition holds.

Proposition 3.4

$$A[\Gamma] \underset{G}{\overset{*}{\Longrightarrow}} w A_1[l_{1,1} \ldots l_{m_1,1}] \ldots A_i[\Gamma l_{1,i} \ldots l_{m_i,i}] \ldots A_n[l_{1,n} \ldots l_{m_n,n}]$$

in a leftmost fashion if and only if

$$\langle q, \ddagger^{\#}_{\#}\Gamma A, \epsilon, w\rangle \models^{*}_{M} \langle q, \ddagger^{\#}_{\#} l_{1,n} \ldots l_{m_n,n} A_n \ldots \ddagger^{\#}_{\#}\Gamma l_{1,i} \ldots l_{m_i,i} A_i \ldots \ddagger l_{1,1} \ldots l_{m_1,1} A_1, w, \epsilon\rangle$$

Given an LIG, G, we define the epda, M, as follows. Each stack in the epda will have a nonterminal symbol on top. Below this nonterminal will be the contents of the stack associated with this nonterminal in the sentential form. Thus, if there are n nonterminals in a sentential form, the epda will have n corresponding stacks. The stack corresponding to the leftmost nonterminal will be the top stack. Thus,

- If $A[\cdots] \rightarrow A_1[] \ldots A_i[l, \cdots] \ldots A_N[]$ is a production in LIG then the epda will have the following move

  $$(q, \ddagger^{\#}_{\#} A_n \ldots \ddagger^{\#}_{\#} A_{i+1}, l A_i, \ddagger^{\#}_{\#} A_{i-1} \ldots \ddagger^{\#}_{\#} A_1) \in \delta(q, \epsilon, A)$$

  Thus, the stack associated with A is passed on to A_i.

- If $A[l, \cdots] \rightarrow A_1[] \ldots A_i[\cdots] \ldots A_n[]$ is a production in LIG then the epda will have the following moves

$$(q_{<l.A_1 \ldots A_i^* \ldots A_n>}, \epsilon, \epsilon, \epsilon) \in (q, \epsilon, A)$$

where

$$(q, \ddagger_2^* A_n \ldots \ddagger_2^* A_{i+1}, A_i, \ddagger_2^* A_{i-1} \ldots \ddagger_2^* A_1) \in \delta(q_{<l.A_1 \ldots A_i^* \ldots A_n}, \epsilon, l >)$$

The pop production is simulated in two steps. In the epda, we have to check if l is immediately below the nonterminal A. Thus, we first remove the top symbol A, record the righthand side of the production in the state, and then check if the index on the top of the side is l before creating new stacks. However, since the present top stack has to be passed on to A_i, we mark A_i in the state information.

- $A[] \rightarrow a$ is a production in the LIG. Then the epda will have the following moves in which we pop symbols off the top stack (associated with A) until it becomes empty. Thus,

$$(q_{erase}, \epsilon, \epsilon, \epsilon) \in \delta(q, \epsilon, A),$$

$$(q_{erase}, \epsilon, \epsilon, \epsilon) \in \delta(q_{erase}, \epsilon, l)$$

$$(q, \epsilon, \epsilon, \epsilon) \in \delta(q_{erase}, a, z)$$

- We include $(q, \epsilon, zS, \epsilon) \in \delta(q, \epsilon, \sigma_0)$ where σ_0 ($\sigma_0 \notin$ nonterminal set of the LIG) is the starting symbol on the store of the epda.

Lemma 3.3 $\quad L(G) \subseteq N(M)$

To prove this lemma, we show by induction that for all j, if

$$A[\Gamma] \underset{lm}{\overset{j}{\Longrightarrow}} wA_1[l_{1,1} \ldots l_{m_1,1}] \ldots A_i[l_{1,i} \ldots l_{m_i,i}] \ldots A_n[l_{1,n} \ldots l_{m_n,n}]$$

then $\langle q, \ddagger\ddagger\Gamma A, \epsilon, w\rangle \models^* \langle q, \ddagger\ddagger l_{1,n}\ldots l_{m_n,n}A_n\ldots\ddagger\ddagger\Gamma l_{1,i}\ldots l_{m_i,i}A_i\ldots\ddagger l_{1,1}\ldots l_{m_1,1}A_1, w, \epsilon\rangle$.

The base case corresponds to $j = 0$ and is obviously true. Let us assume that the above hypothesis holds for all $j < k$ and suppose

$$A[\Gamma] \underset{lm}{\overset{k}{\Longrightarrow}} wA_1[l_{1,1}\ldots l_{m_1,1}]\ldots A_i[\Gamma l_{1,i}\ldots l_{m_i,i}]\ldots A_n[l_{1,n}\ldots l_{m_n,n}]$$

Then several cases arise.

Case 1 In the final step let the leftmost nonterminal be rewritten using a *push* production. Then we can assume that the following $k-1$ step derivation took place.

$$A[\Gamma] \underset{lm}{\overset{k-1}{\Longrightarrow}} wB[l'_1\ldots l'_{n'}]A_q[l_{1,q}\ldots l_{m_q,q}]\ldots A_n[l_{1,n}\ldots l_{m_n,n}]$$

Also let the *push* production used be

$$B[\cdot\cdot] \rightarrow A_1[]\ldots A_{i'}[l',\cdot\cdot]\ldots A_{q-1}[]$$

There are two cases to consider, when $q < i$ or $q \geq i$.

case 1a $q < i$: Since the last production used is

$$B[\cdot\cdot] \rightarrow A_1[]\ldots A_{i'}[l',\cdot\cdot]\ldots A_{q-1}[]$$

it has to be the case that $l' = l_{m_{i'},i'}.l'_1\ldots l'_{n'} = l_{1,i'}\ldots l_{(m_{i'}-1),i'}$ and

$$A[\Gamma] \underset{lm}{\overset{k-1}{\Longrightarrow}} wB[l_{1,i'}\ldots l_{(m_{i'}-1),i'}]A_q[l_{1,q}\ldots l_{m_q,q}]\ldots l_{m_i,i}]\ldots A_n[l_{1,n}\ldots l_{m_n,n}]$$

Thus,

$$A[\Gamma] \underset{lm}{\overset{k}{\Longrightarrow}} wA_1[]\ldots A_{i'}[l_{1,i'}\ldots l_{m_{i'},i'}]\ldots A_{q-1}[]A_q[l_{1,q} \ldots l_{m_q,q}]\ldots A_i[\Gamma l_{1,i}\ldots l_{m_i,i}]\ldots A_n[l_{1,n}\ldots l_{m_n,n}]$$

By inductive hypothesis, we can assume that

$$\langle q, \ddagger\ddagger\Gamma A, \epsilon, w\rangle \models^* \langle q, \ddagger\ddagger l_{1,n}\ldots l_{m_n,n}A_n\ldots\ddagger\ddagger\Gamma l_{1,i}\ldots l_{m_i,i}A_i\ldots\ddagger\ddagger l_{1,q}\ldots l_{m_q,q}A_q \ddagger\ddagger l_{1,i'}\ldots l_{m_{i'-1},i'}B, w, \epsilon\rangle$$

and because of the *push* production used, by the definition of the epda, we have

$$\langle q, \ddagger_i \Gamma A, \epsilon, w\rangle \models^* \langle q, \ddagger_i l_{1,n} \ldots l_{m_n,n} A_n \ldots \ddagger_i \Gamma l_{1,i} \ldots l_{m_i,i} A_i \ldots \ddagger_i l_{1,q} \ldots l_{m_q,q} A_q$$
$$\ddagger_i A_{q-1} \ldots \ddagger_i l_{1,i'} \ldots l_{m_{i'},i'} A_{i'} \ldots \ddagger_i A_1, w, \epsilon\rangle$$

as required.

case 2 $q \geq i$: Since the last production used is

$$B[\cdot\cdot] \rightarrow A_1[] \ldots A_{i'}[l', \cdot\cdot] \ldots A_{q-1}[]$$

it has to be the case that $l' = l_{m_i,i}, l'_1 \ldots l'_{n'} = l_{1,i} \ldots l_{(m_i-1),i}, A'_i = A_i$ and

$$A[\Gamma] \underset{lm}{\overset{k-1}{\Longrightarrow}} wB[l_{1,i} \ldots l_{(m_i-1),i}] A_q[l_{1,q} \ldots l_{m_q,q}] \ldots A_n[l_{1,n} \ldots l_{m_n,n}]$$

Thus,

$$A[\Gamma] \underset{lm}{\overset{k}{\Longrightarrow}} wA_1[] \ldots A_i[l_{1,i} \ldots l_{m_i,i}] \ldots A_{q-1}[] A_q[l_{1,q} \ldots l_{m_q,q}] \ldots A_n[l_{1,n} \ldots l_{m_n,n}]$$

By inductive hypothesis, we can assume that

$$\langle q, \ddagger_i \Gamma A, \epsilon, w\rangle \models^* \langle q, \ddagger_i l_{1,n} \ldots l_{m_n,n} A_n \ldots \ddagger_i l_{1,q} \ldots l_{m_q,q} A_q \ddagger_i l_{1,i} \ldots l_{m_i-1,i} B, w, \epsilon\rangle$$

and because of the *push* production used, by the definition of the epda, we have

$$\langle q, \ddagger_a \Gamma A, \epsilon, w\rangle \models^* \langle q, \ddagger_a l_{1,n} \ldots l_{m_n,n} A_n \ldots \ddagger_a l_{1,q} \ldots l_{m_q,q} A_q$$
$$\ddagger_a A_{q-1} \ldots \ddagger_a l_{1,i} \ldots l_{m_i,i} A_i \ldots \ddagger_a A_1, w, \epsilon\rangle$$

as required.

Case 2: In the last step of the derivation, let the leftmost nonterminal be rewritten using a *pop* production. Then we can assume that the following $k-1$ step derivation took place.

$$A[\Gamma] \underset{lm}{\overset{k-1}{\Longrightarrow}} wB[l'_1 \ldots l'_{n'}] A_q[l_{1,q} \ldots l_{m_q,q}] \ldots A_n[l_{1,n} \ldots l_{m_n,n}]$$

Also let the *pop* production used be

$$B[l', \cdot\cdot] \rightarrow A_1[] \ldots A_{i'}[\cdot\cdot] \ldots A_{q-1}[]$$

There are two cases to consider, when $q < i$ or $q \geq i$.

case 2a $q < i$: Since the last production used is

$$B[l', \cdot\cdot] \rightarrow A_1[] \ldots A_{i'}[\cdot\cdot] \ldots A_{q-1}[]$$

it has to be the case that $l' = l_{m_{i'}+1,i'}, l'_1 \ldots l'_{n'} = l_{1,i'} \ldots l_{(m_{i'}+1),i'}$ and

$$A[\Gamma] \underset{lm}{\overset{k-1}{\Longrightarrow}} wB[l_{1,i'} \ldots l_{(m_{i'}+1),i'}]A_q[l_{1,q} \ldots l_{m_q,q}] \ldots A_i[l_{1,i} \ldots l_{m_i,i}] \ldots A_n[l_{1,n} \ldots l_{m_n,n}]$$

Thus,

$$A[\Gamma] \underset{lm}{\overset{k}{\Longrightarrow}} wA_1[] \ldots A_{i'}[l_{1,i'} \ldots l_{m_{i'},i'}] \ldots A_{q-1}[]$$
$$A_q[l_{1,q} \ldots l_{m_q,q}] \ldots A_i[\Gamma l_{1,i} \ldots l_{m_i,i}] \ldots A_n[l_{1,n} \ldots l_{m_n,n}]$$

By inductive hypothesis, we can assume that

$$\langle q, \ddagger\ddagger\Gamma A, \epsilon, w\rangle \models^* \langle q, \ddagger\ddagger l_{1,n} \ldots l_{m_n,n}A_n \ldots \ddagger\ddagger\Gamma l_{1,i} \ldots l_{m_i,i}A_i \ldots \ddagger\ddagger l_{1,q} \ldots l_{m_q,q}A_q$$
$$\ddagger\ddagger l_{1,i'} \ldots l_{m_{i'}+1,i'}B, w, \epsilon\rangle$$

and because of the *pop* production used, by the definition of the epda, we know that

$$(q_{<l',A_1,\ldots,A'_{i'},\ldots,A_{q-1}>}, \epsilon, \epsilon, \epsilon) \in \delta(q, \epsilon, B)$$

and

$$(q, \ddagger\ddagger A_{q-1} \ldots\ldots \ddagger\ddagger A_{i'+1}, A_{i'}, \ddagger\ddagger A_{i'-1} \ldots \ddagger\ddagger A_1) \in \delta(q_{<l',A_1,\ldots,A'_{i'},\ldots,A_{q-1}>}, \epsilon, \epsilon)$$

Thus, we have

$$\langle q, \ddagger\ddagger\Gamma A, \epsilon, w\rangle \models^* \langle q, \ddagger\ddagger l_{1,n} \ldots l_{m_n,n}A_n \ldots \ddagger\ddagger\Gamma l_{1,i} \ldots l_{m_i,i}A_i \ldots \ddagger\ddagger l_{1,q} \ldots l_{m_q,q}A_q$$
$$\ddagger\ddagger A_{q-1} \ldots \ddagger\ddagger l_{1,i'} \ldots l_{m_{i'},i'}A_{i'} \ldots \ddagger\ddagger A_1, w, \epsilon\rangle \text{ as required.}$$

case 2b $q \geq i$: Since the last production used is

$$B[\cdot\cdot] \rightarrow A_1[] \ldots A_{i'}[l', \cdot\cdot] \ldots A_{q-1}[]$$

it has to be the case that $l' = l_{m_i+1,i}, l'_1 \ldots l'_{n'} = l_{1,i} \ldots l_{(m_i+1),i}, A'_i = A_i$ and

$$A[\Gamma] \underset{lm}{\overset{k-1}{\Longrightarrow}} wB[l_{1,i} \ldots l_{(m_i+1),i}]A_q[l_{1,q} \ldots l_{m_q,q}] \ldots A_n[l_{1,n} \ldots l_{m_n,n}]$$

Thus,

$$A[\Gamma] \underset{lm}{\overset{k}{\Longrightarrow}} wA_1[] \ldots A_i[l_{1,i} \ldots l_{m_i,i}] \ldots A_{q-1}[]A_q[l_{1,q} \ldots l_{m_q,q}] \ldots A_n[l_{1,n} \ldots l_{m_n,n}]$$

By inductive hypothesis, we can assume that

$$\langle q, \ddagger_{\sigma}^{\sigma}\Gamma A, \epsilon, w\rangle \models^{*} \langle q, \ddagger_{\sigma}^{\sigma} l_{1,n} \ldots l_{m_n,n} A_n \ldots \ddagger_{\sigma}^{\sigma} l_{1,q} \ldots l_{m_q,q} A_q \ddagger_{\sigma}^{\sigma} l_{1,i} \ldots l_{m_i+1,i} B, w, \epsilon\rangle$$

and because of the *push* production used, by the definition of the epda, we have

$$\langle q, \ddagger_{\sigma}^{\sigma}\Gamma A, \epsilon, w\rangle \models^{*} \langle q, \ddagger_{\sigma}^{\sigma} l_{1,n} \ldots l_{m_n,n} A_n \ldots \ddagger_{\sigma}^{\sigma} l_{1,q} \ldots l_{m_q,q} A_q \ddagger_{\sigma}^{\sigma} A_{q-1} \ldots \ddagger_{\sigma}^{\sigma} l_{1,i} \ldots l_{m_i,i} A_i \ldots \ddagger_{\sigma}^{\sigma} A_1, w, \epsilon\rangle$$

as required.

Case 3: The leftmost nonterminal was rewritten using a *terminal* production. Let this production be $B[\cdot\cdot] \rightarrow \varepsilon$. Then if

$$A[\Gamma] \underset{lm}{\overset{k}{\Longrightarrow}} w A_1[l_{1,1} \ldots l_{m_1,1}] \ldots A_i[\Gamma l_{1,i} \ldots l_{m_i,i}] \ldots A_n[l_{1,n} \ldots l_{m_n,n}]$$

then it has to be the case that

$$A[\Gamma] \underset{lm}{\overset{k-1}{\Longrightarrow}} w_1 B[\Gamma] A_1[l_{1,1} \ldots l_{m_1,1}] \ldots A_i[\Gamma l_{1,i} \ldots l_{m_i,i}] \ldots A_n[l_{1,n} \ldots l_{m_n,n}]$$

, where $w_1\varepsilon = w$. By the inductive hypothesis, we can assume

$$\langle q, \ddagger_{\sigma}^{\sigma}\Gamma A, \epsilon, w_1\rangle \models^{*} \langle q, \ddagger_{\sigma}^{\sigma} l_{1,n} \ldots l_{m_n,n} A_n \ldots \ddagger_{\sigma}^{\sigma} l_{1,1} \ldots l_{m_1,1} A_1 \ddagger_{\sigma}^{\sigma}\Gamma B, w_1, \epsilon\rangle$$

Because of the production used, we will have $(q_{erase}, \epsilon, \epsilon, \epsilon) \in \delta(q, \epsilon, B)$, $(q_{erase}, \epsilon, \epsilon, \epsilon) \in \delta(q_{erase}, \epsilon, \gamma)$ $(\gamma \neq z)$ and $(q, \epsilon, \epsilon, \epsilon) \in \delta(q_{erase}, \varepsilon, z)$ and thus we see the inductive statement used holds for this case too.

To show the inclusion of the language generated by the LIG in the language accepted by the epda, we have to consider a derivation of w from $S[]$ and from the initial configuration of $\langle q, \ddagger\sigma_0, \epsilon, w\rangle$ from which we can generate the ID $\langle q, \ddagger S, \epsilon, w\rangle$. To show the inclusion in the other direction, we have to use the implication in the other direction of the proposition given above. Thus the induction hypothesis is that

$$A[\Gamma] \underset{lm}{\overset{*}{\Longrightarrow}} w A_1[l_{1,1} \ldots l_{m_1,1}] \ldots A_i[\Gamma l_{1,i} \ldots l_{m_i,i}] \ldots A_n[l_{1,n} \ldots l_{m_n,n}]$$

only if

$$\langle q, \ddagger\ddagger\Gamma A, \epsilon, w\rangle \models^k \langle q, \ddagger\ddagger l_{1,n}\ldots l_{m_n,n}A_n\ldots\ddagger\ddagger\Gamma l_{1,i}\ldots l_{m_i,i}A_i\ldots\ddagger l_{1,1}\ldots l_{m_1,1}A_1, w, \epsilon\rangle$$

We assume that this induction hypothesis holds when the epda makes less than k moves. The same 5 cases (but as moves of the epda) will arise and can be dealt with in similar fashion. For example, let the following computation take place.

$$\langle q, \ddagger\ddagger\Gamma A, \epsilon, w\rangle \models^k \langle q, \ddagger\ddagger l_{1,n}\ldots l_{m_n,n}A_n\ldots\ddagger\ddagger\Gamma l_{1,i}\ldots l_{m_i,i}A_i\ldots\ddagger l_{1,1}\ldots l_{m_1,1}A_1, w, \epsilon\rangle$$

Then depending on the last move of the epda one of the possible computations would be

$\langle q, \ddagger\ddagger\Gamma A, \epsilon, w\rangle \models^{k-1} \langle q, \ddagger\ddagger l_{1,n}\ldots l_{m_n,n}A_n\ldots\ddagger\ddagger l_{1,q}\ldots l_{m_q,q}A_q\ddagger\ddagger l'_1\ldots l'_n B, w, \epsilon\rangle$

$\models \langle q, \ddagger\ddagger l_{1,n}\ldots l_{m_n,n}A_n\ldots\ddagger\ddagger\Gamma l_{1,i}\ldots l_{m_i,i}A_i\ldots\ddagger\ddagger l_{1,q}\ldots l_{m_q,q}A_q$

$\ddagger\ddagger A_{q-1}\ldots\ddagger l_{1,i'}\ldots l_{m_{i'},i'}A_{i'}\ldots\ddagger\ddagger A_1, w, \epsilon\rangle$

where $l'_1\ldots l'_{n'} = l_{1,i'}\ldots l_{m_{i'}-1,i'}$. Therefore the LIG will have the production

$$B[\cdot\cdot] \rightarrow A_1[]\ldots A_i[l_{m_{i'},i'}, \cdot\cdot]\ldots A_{q-1}$$

. With the use of this production and the inductive hypothesis, we get

$$A[\Gamma] \underset{lm}{\Longrightarrow} wB[l_{1,i'}\ldots l_{m_{i'}-1,i'}]A_q[l_{1,q}\ldots l_{m_q,q}]\ldots A_i[\Gamma l_{1,i}\ldots l_{m_i,i}]\ldots A_1[l_{1,1}\ldots l_{m_1,1}]$$

as required. The other four cases can be similarly worked out.

So far, we have seen that for every LIG there is an epda which accepts the language generated by the LIG. Hence the class of LIL's are included in the class of TAL's. Now we will show the inclusion of TAL's in LIL's. We can show that for every epda there is a LIG which generates the language accepted by the epda. The LIG constructed will satisfy a similar proposition as considered above. However, as in the case of constructing a MHG from an epda, we will consider nonterminals which will have 4 states of the epda in its name. Again, there will be several cases to consider. Since we have already gone through that exercise in constructing a MHG from an epda, we can directly convert an MHG to a LIG, and thus show the inclusion of TAL's in LIL's.

3.3.3 Inclusion of TAL's in LIL's

Theorem 3.7 For every MHG, $G = (N, \Sigma, P, S)$, there is a LIG, G', such that $L(G) = L(G')$.

The nonterminals used in G' are just S' (the start symbol), X and $X_\sharp$ ($X, X_\sharp \notin N$). The set of indices used in G' are the nonterminals of G and a special symbol $\sharp$ which does not belong to N. $\sharp$ and $X_\sharp$ are used to detect the bottom of the stack associated with the nonterminals in G'. The proposition that G and G' satisfy is as follows:

Proposition 3.5

$A \overset{*}{\underset{G}{\Rightarrow}} w_1 \gamma w_2$ if and only if $X[\Gamma A] \overset{*}{\underset{G'}{\Rightarrow}} w_1 X[\Gamma] \gamma w_2$

1. $S'[\cdot\cdot] \rightarrow X[\sharp, \cdot\cdot]$
2. $X[\sharp, \cdot\cdot] \rightarrow X_\sharp[\cdot\cdot]$ The bottom of the stack has been reached.
3. $X_\sharp[] \rightarrow \epsilon$ Any nonterminal of G, which is to be rewritten, will be represented on the stack. Thus, if we have reached the bottom of the stack, which is denoted by the nonterminal $X_\sharp$, we remove it.
4. $A \rightarrow W(B, C)$ if and only if $X[A, \cdot\cdot] \rightarrow X[CB, \cdot\cdot]$
5. $A \rightarrow C1(B, C)$ if and only if $X[A, \cdot\cdot] \rightarrow X[B, \cdot\cdot]X[C\sharp]$
6. $A \rightarrow C2(B, C)$ if and only if $X[A, \cdot\cdot] \rightarrow X[B\sharp]X[C, \cdot\cdot]$
7. $A \rightarrow \gamma\varepsilon$ if and only if $X[A, \cdot\cdot] \rightarrow \varepsilon X[\cdot\cdot]$ where $\varepsilon \in \Sigma \cup \{\epsilon\}$
8. $A \rightarrow \varepsilon\gamma$ if and only if $X[A, \cdot\cdot] \rightarrow X[\cdot\cdot]\varepsilon$

Lemma 3.4 $L(G) \subseteq L(G')$

To show this inclusion, we assume that for all $k_1 < k$ the following holds:

If $A \overset{k_1}{\underset{G}{\Longrightarrow}} w_1 \uparrow w_2$ then $X[\Gamma A] \overset{*}{\underset{G'}{\Longrightarrow}} w_1 X[\Gamma] w_2$ (I)

The base case corresponds to $k = 1$ and an application of the production of the form $A \rightarrow \uparrow\varepsilon$ or $A \rightarrow \varepsilon\uparrow$. Then the production $X[A, \cdot\cdot] \rightarrow X[\cdot\cdot]\varepsilon$ or $X[A, \cdot\cdot] \rightarrow \varepsilon X[\cdot\cdot]$ respectively will be in G' and thus the base case holds.

For the induction case, let us say $A \overset{k}{\underset{G}{\Longrightarrow}} w_1 \uparrow w_2$. Then one of the following holds.

1. $A \rightarrow W(B, C)$, $B \overset{k_1}{\underset{G}{\Longrightarrow}} u_1 \uparrow u_2$, and $C \overset{k_2}{\underset{G}{\Longrightarrow}} v_1 \uparrow v_2$. Then $w_1 = u_1 v_1$, $w_2 = v_2 u_2$, and $k = k_1 + k_2 + 1$. Hence, the inductive hypothesis holds for B and C. Thus, $X[\Gamma_1 B] \overset{*}{\underset{G'}{\Longrightarrow}} u_1 X[\Gamma_1] u_2$ and $X[\Gamma C] \overset{*}{\underset{G'}{\Longrightarrow}} v_1 X[\Gamma] v_2$ for all Γ_1, Γ. Also, $X[A, \cdot\cdot] \rightarrow X[BC, \cdot\cdot]$. Thus, $X[\Gamma A] \overset{1}{\underset{G'}{\Longrightarrow}} X[\Gamma C B] \overset{*}{\underset{G'}{\Longrightarrow}} u_1 X[\Gamma C] u_2 \overset{*}{\underset{G'}{\Longrightarrow}} u_1 v_1 X[\Gamma] v_2 u_2$ as required.

2. $A \rightarrow C1(B, C)$, $B \overset{k_1}{\underset{G}{\Longrightarrow}} u_1 \uparrow u_2$, and $C \overset{k_2}{\underset{G}{\Longrightarrow}} v_1 \uparrow v_2$. Then $w_1 = u_1$, $w_2 = u_2 v_1 v_2$, and $k = k_1 + k_2 + 1$. Hence, the inductive hypothesis holds for B and C. Thus, $X[\Gamma_1 B] \overset{*}{\underset{G'}{\Longrightarrow}} u_1 X[\Gamma_1] u_2$ and $X[\Gamma C] \overset{*}{\underset{G'}{\Longrightarrow}} v_1 X[\Gamma] v_2$ for all Γ_1, Γ. Also, $X[A, \cdot\cdot] \rightarrow X[B, \cdot\cdot] X[\uparrow C]$. Thus, $X[\Gamma A] \overset{1}{\underset{G'}{\Longrightarrow}} X[\Gamma B] X[\uparrow C] \overset{*}{\underset{G'}{\Longrightarrow}} u_1 X[\Gamma] u_2 X[\uparrow C]$ $\overset{*}{\underset{G'}{\Longrightarrow}} u_1 X[\Gamma] u_2 v_1 X[\uparrow] v_2 \overset{*}{\underset{G'}{\Longrightarrow}} u_1 X[\Gamma] u_2 v_1 v_2$ as required.

3. $A \rightarrow C2(B, C)$, $B \overset{k_1}{\underset{G}{\Longrightarrow}} u_1 \uparrow u_2$, and $C \overset{k_2}{\underset{G}{\Longrightarrow}} v_1 \uparrow v_2$. Then $w_1 = u_1 u_2 v_1$, $w_2 = v_2$, and $k = k_1 + k_2 + 1$. Hence, the inductive hypothesis holds for B and C. Thus, $X[\Gamma_1 B] \overset{*}{\underset{G'}{\Longrightarrow}} u_1 X[\Gamma_1] u_2$ and $X[\Gamma C] \overset{*}{\underset{G'}{\Longrightarrow}} v_1 X[\Gamma] v_2$ for all Γ_1, Γ. Also, $X[A, \cdot\cdot] \rightarrow X[\uparrow B] X[C, \cdot\cdot]$. Thus, $X[\Gamma A] \overset{*}{\underset{G'}{\Longrightarrow}} u_1 X[\uparrow] u_2 v_1 X[\Gamma] v_2 \overset{*}{\underset{G'}{\Longrightarrow}}$ $u_1 u_2 v_1 X[\Gamma] v_2$ as required.

If $S[] \overset{*}{\underset{G}{\Longrightarrow}} w_1 \uparrow w_2$, then $X[\uparrow S] \overset{*}{\underset{G'}{\Longrightarrow}} w_1 X[\uparrow] w_2 \overset{*}{\underset{G'}{\Longrightarrow}} w_1 w_2$. Since, we have the production, $S' \rightarrow X[\uparrow S]$, we have $L(G) \subseteq L(G')$.

Lemma 3.5 $L(G') \subseteq L(G)$

To prove this lemma, we inductively assume that if $k_1 < k$ then $A \overset{*}{\underset{G}{\Longrightarrow}} w_1 w_2$ only if $X[\Gamma A] \overset{k_1}{\underset{G'}{\Longrightarrow}} w_1 X[\Gamma] w_2$.

The base case

Suppose $X[\Gamma A] \overset{k}{\underset{G'}{\Longrightarrow}} w_1 X[\Gamma] w_2$. Then one of the following case holds.

1. $X[A, \cdot\cdot] \rightarrow X[CB, \cdot\cdot]$, $X[\Gamma_1 B] \overset{k_1}{\underset{G'}{\Longrightarrow}} u_1 X[\Gamma_1] u_2$ $X[\Gamma_2 C] \overset{k_2}{\underset{G'}{\Longrightarrow}} v_1 X[\Gamma_2] v_2$ where $\Gamma_1 = \Gamma C, \Gamma_2 = \Gamma$, $w_1 = u_1 v_1$, $w_2 = v_2 u_2$, and $k = k_1 + k_2 + 1$. Then the inductive hypothesis holds for B, C. Thus, $B \overset{*}{\underset{G}{\Longrightarrow}} u_1 u_2$ and $C \overset{*}{\underset{G}{\Longrightarrow}} v_1 v_2$. Since, $X[A, \cdot\cdot] \rightarrow X[CB, \cdot\cdot]$, $A \rightarrow W(B, C)$. Thus, $A \overset{*}{\underset{G}{\Longrightarrow}} W(B, C) \overset{*}{\underset{G}{\Longrightarrow}} u_1 v_1 \cdot v_2 u_2$ as required.

2. $X[A, \cdot\cdot] \rightarrow X[CB, \cdot\cdot]$, $X[\Gamma_1 B] \overset{k_1}{\underset{G'}{\Longrightarrow}} u_1 X[\Gamma_1] u_2$ $X[\Gamma_2 C] \overset{k_2}{\underset{G'}{\Longrightarrow}} v_1 X[\Gamma_2] v_2$ where $\Gamma_1 = \Gamma, \Gamma_2 = \varepsilon$, $w_1 = u_1 u_2 v_1$, $w_2 = v_2$, and $k = k_1 + k_2 + 1$. Then the inductive hypothesis holds for B, C. Thus, $B \overset{*}{\underset{G}{\Longrightarrow}} u_1 u_2$ and $C \overset{*}{\underset{G}{\Longrightarrow}} v_1 v_2$. Since, $X[A, \cdot\cdot] \rightarrow X[B, \cdot\cdot] X[\natural C]$, $A \rightarrow C1(B, C)$. Thus, $A \overset{*}{\underset{G}{\Longrightarrow}} C1(B, C) \overset{*}{\underset{G}{\Longrightarrow}} u_1 u_2 v_1 v_2$ as required.

3. $X[A, \cdot\cdot] \rightarrow X[\natural B] X[C, \cdot\cdot]$, $X[\Gamma_1 B] \overset{k_1}{\underset{G'}{\Longrightarrow}} u_1 X[\Gamma_1] u_2$ $X[\Gamma_2 C] \overset{k_2}{\underset{G'}{\Longrightarrow}} v_1 X[\Gamma_2] v_2$ where $\Gamma_1 = \varepsilon, \Gamma_2 = \Gamma$, $w_1 = u_1 u_2 v_1$, $w_2 = v_2$, and $k = k_1 + k_2 + 1$. Then the inductive hypothesis holds for B, C. Thus, $B \overset{*}{\underset{G}{\Longrightarrow}} u_1 u_2$ and $C \overset{*}{\underset{G}{\Longrightarrow}} v_1 v_2$. Since, $X[A, \cdot\cdot] \rightarrow X[\natural B] X[C, \cdot\cdot]$, $A \rightarrow C2(B, C)$. Thus, $A \overset{*}{\underset{G}{\Longrightarrow}} C2(B, C) \overset{*}{\underset{G}{\Longrightarrow}} u_1 u_2 v_1 v_2$ as required.

Thus, if $X[\natural S] \overset{*}{\underset{G'}{\Longrightarrow}} w_1 X[\natural] w_2 \overset{*}{\underset{G'}{\Longrightarrow}} w_1 w_2$, then $S \overset{*}{\underset{G'}{\Longrightarrow}} w_1 w_2$. Since we have the production, $S'[\cdot\cdot] \rightarrow X[\natural S]$, we have shown $L(G') \subseteq L(G)$.

Thus, we have $L(G) = L(G')$.

Chapter 4

Other Results About TAL's

In this chapter, we establish some results about TAL's. First we examine the closure properties of TAL's and show that TAL's form an *Abstract Family of Languages*. In addition to their closure results, we characterize TAL's by giving a pumping lemma, similar to the $uvwxy$ theorem for CFL's. Another characterization of TAL's is given by showing that every TAL is a semilinear language and therefore has the *constant-growth* property. There has been some interest in showing the above properties for grammatical systems for natural languages (see [Joshi 85,Berwick 84]). Finally, we show that parsing of TAL's can be done in polynomial time.

4.1 Closure Properties of Tree Adjoining Languages

In this chapter, we shall study the closure properties for TAL's. Some of these properties have been proved in [VijayShanker 85]. We shall show that the class of Tree Adjoining Languages, like the class of Context-free Languages, form an substitution closed Full *Abstract Family of Languages* (AFL). Thus, they are closed under the set operations of union, concatenation, Kleene-star, intersection with Regular Languages, homomorphism, inverse homomorphism, and substitution. In this chapter, we prove these closure results (except for the closure under *intersection with Regular Languages* which is proved in Section 3.2.3). Also, using a pumping lemma for TAL's, which we shall prove in Section 4.2, we show that TAL's are not closed under *intersection*, and *intersection with CFL's*.

Theorem 4.1 TAL's are closed under union.

Let L_1 and L_2 be TAL's. Let G_1 and G_2 be TAG's such that $L_1 = L(G_1)$ and $L_2 = L(G_2)$. Let $G_i = (N_i, \Sigma_i, S_i, I_i, A_i)$ for $i \in \{1,2\}$. Without loss of generality, we may assume that $N_1 \cap N_2 = \phi$. Further, we also assume that $I_i, i \in \{1,2\}$ contain one initial tree, α_i, each of the form described in Section 1.15.

We can give a TAG, $G = (N_1 \cup N_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, S, I, A)$ (where $S \notin N_1 \cup N_2$), such that $L(G) = L_1 \cup L_2$. G is obtained as follows. We let $A = A_1 \cup A_2$. I contains two initial trees, α_{I_1} and α_{I_2}, which are given in Figure 4.1. Showing $L(G) = L_1 \cup L_2$ is straightforward.

Theorem 4.2 TAL's are closed under concatenation.

Let L_1 and L_2 be TAL's. Let G_1 and G_2 be TAG's such that $L_1 = L(G_1)$ and $L_2 = L(G_2)$. Let $G_i = (N_i, \Sigma_i, S_i, I_i, A_i)$ for $i \in \{1,2\}$. Without loss of generality, we

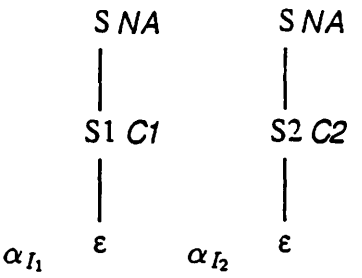

Figure 4.1: Closure under union

may assume that $N_1 \cap N_2 = \phi$. Further, we also assume that $I_i, i \in \{1,2\}$ contain one initial tree each of the form described in Section 1.15.

We can give a TAG, $G = (N_1 \cup N_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, S, I, A)$ (where $S \notin N_1 \cup N_2$), such that $L(G) = L_1 \cdot L_2$. G is obtained as follows. We let $A = A_1 \cup A_2$. I contains one initial tree which is given in Figure 4.2.

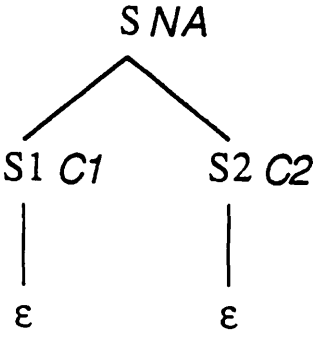

Figure 4.2: Closure under concatenation

Showing $L(G) = L_1 \cdot L_2$ is straightforward.

Theorem 4.3 TAL's are closed under Kleene closure.

Let L_1 be a TAL. Let $G_1 = (N_1, \Sigma_1, S_1, I_1, A_1)$ be a TAG such that $L_1 = L(G_1)$. We assume that I_1 contains one initial tree of the form described in Section 1.15.

We can give a TAG, $G = (N_1 \cup \{S\}, \Sigma_1, S, I, A)$ (where $S \notin N_1$), such that $L(G) = L_1^*$. G is obtained as follows. We let $A = A_1 \cup \{\beta\}$. I contains one initial tree α. The elementary trees α and β are given in Figure 4.3. Showing $L(G) = L_1^*$ is straightforward.

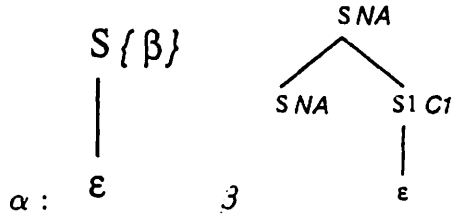

Figure 4.3: Closure under Kleene closure

Theorem 4.4 TAL's are closed under substitution.

Let L be a Tree Adjoining Language and let the TAG, $G = (N, \Sigma, I, A, S)$ generate L. Let τ be a substitution such that $\tau(a) = L_a$. Let $G_a = (N_a, \Sigma_a, I_a, A_a, S_a)$ be a TAG generating L_a. We want to show that $\tau(L)$ is a TAL. Let G' be a TAG whose elementary trees are given by the following construction. We show that G' generates $\tau(L(G))$.

For each elementary tree, $\gamma \in I \cup A$, all nodes in frontier of γ labelled by the terminal a, are replaced by an initial tree $\alpha_a \in I_a$.

We can show that G' indeed generates $\tau(L(G)) = L$.

Corollary 4.1 TAL's are closed under arbitrary homomorphism.

Theorem 4.5 TAL's are closed under intersection with Regular Languages.

In Chapter 3, we describe an automata called the *Embedded Pushdown Automata*, an extension of the Pushdown Automata, and show they recognize exactly TAL's. We can

extend the proof which uses the pda to show that CFL's are closed under intersection with Regular Languages to a proof using the epda instead for this theorem. The proof is given in Section 3.2.3.

Corollary 4.2 TAL's are closed under inverse homomorphism.

The class of regular languages is contained in the class of Tree Adjoining Languages. Thus, TAL's are closed under regular substitution. Using a theorem in [Ginsburg 66, pages 121-122], we have closure under inverse homomorphism.

Thus, we have shown that TAL's form a *substitution closed Full Abstract Family of Languages* since we have shown that they are closed under the operations of union, concatenation, Kleene-star, homomorphism, inverse homomorphism, intersection with Regular Languages and substitution. We know turn our attention to showing some non-closure results for TAL's.

Theorem 4.6 TAL's are not closed under intersection or intersection with CFL's and complementation.

In Section 4.2, we give a pumping lemma for TAL's and show $\{a^n b^n c^n d^n e^n \mid n \geq 0\}$ is not a TAL. This allows us to prove that TAL's are not closed under intersection or intersection with CFL's. It can be shown that the languages $\{a^n b^n c^n d^n e^m \mid m, n \geq 0\}$ and $\{a^{m_1} b^{m_2} c^{m_3} d^n e^n \mid m_1, m_2, m_3, n \geq 0\}$ are TAL's. The latter language is a CFL too. The intersection of these two languages is $\{a^n b^n c^n d^n e^n \mid n \geq 0\}$ which is not a TAL. Thus, TAL's are not closed under intersection with CFL's or intersection with TAL's. Since TAL's are closed under union and not closed under intersection, TAL's are not closed under complementation.

4.2 Pumping Lemma for TAL's

We now give a pumping lemma for TAL's, adapting the $uvwxy$-theorem for CFL's. In a CFG, generating an infinite language, any derivation of a string of length more than a fixed number must involve the use of a nonterminal, X, such that $X \stackrel{+}{\underset{G}{\Longrightarrow}} w_1 X w_2$. Hence this subderivation may used any number of times, leading to the derivation of arbitrarily long strings. In a TAG, G, the derivation trees (given by $T_D(G)$) are similar to derivation trees in CFG, and hence it suggests that if the language derived by G is infinite, then so must $T_D(G)$. Just as in CFG's, there will be a derived auxiliary trees which can be adjoined with itself. Thus, in TAL's, derived auxiliary trees play the role in a recursive derivation which nonterminals do in CFG's.

To discuss the derivations in a TAG, we have to consider the derivation trees. We will first show that if there is a sufficiently large derivation tree, then there are arbitrarily large derivation trees. We will then consider the derivation of strings (in the frontier of the derived trees described by these derivation trees).

Without loss of generality, we will assume that the only elementary trees containing a terminal symbol in the frontier are auxiliary trees of the following form. This assumption

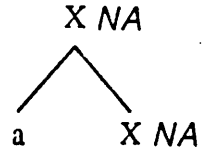

Figure 4.4: Auxiliary trees with terminal symbols

can be made because if there is any other form of elementary tree which has a terminal symbol "a" in the frontier then that frontier node is replaced by a subtree of the form given in 4.5, where β_a is also given in 4.5. Once we have made this assumption, we can assume that if γ is a derived tree with the frontier containing k terminal symbols, then

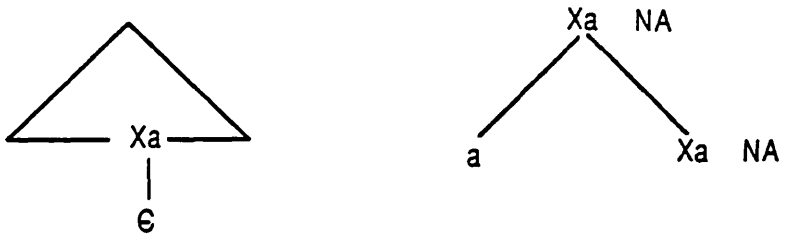

Figure 4.5: The auxiliary tree $beta_a$

every derivation tree that describes γ will have exactly k occurences of nodes labelled β_a ($a \in \Sigma$).

Let us assume that the maximum length of the right-hand side of a derivation rule is m. Let us also assume that the number of elementary trees is k. First we will show that if there is a derivation tree, Υ, with no path of length greater than i, then the number of occurences, j, of subtrees of the form given in figure 4.5 is not greater than m^{i-1}. We shall prove this result by induction. Base Case: $i = 1$
Then the derivation tree is of the form given below. Therefore $j = 1$ if $\gamma = \beta_a$ for some

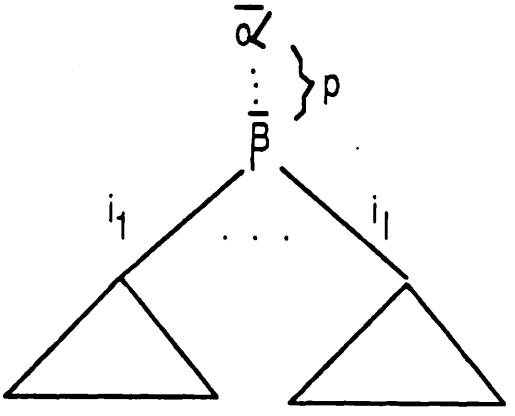

Figure 4.6: Derivation tree of height one

$a \in \Sigma$ else $j = 0$.

Inductive Case: The derivation tree is of the form given below. No path in $T_1 \ldots . T_l$ is of length greater than $i - p - 1$. Thus, by inductive hypothesis, $j_q \leq m^{(i-p-2)}$ for $1 \leq q \leq l$. Thus, $j \leq l \cdot m^{(i-p-2)} \leq m^{(i-p-1)} \leq m^{(i-1)}$ as required.

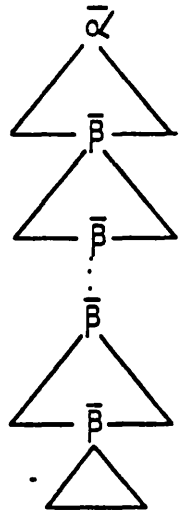

Figure 4.7: Derivation tree of height i

Now consider the derivation of a tree whose frontier is the string $z \in \Sigma$. Let $|z| \geq n$ where $n = m^k$. Every derivation tree which describes a derived tree with frontier z must have $|z|$ occurences of nodes in the frontier labeled β_a for some $a \in \Sigma$. Then, these derivation trees must have a path of length at least $k + 2$. Leaving aside the node in the frontier, there must be at least $k + 1$ vertices, and hence for some auxiliary tree, β, two nodes in this path must be labeled by $\overline{\beta}$. Let the derivation tree be represented as in figure 4.8. Consider the string $v_2'v_4'$. If it does not have at least one occurence of β_a as a substring, then consider the derivation tree Υ'' obtained by removing the subtree with $v_2'v_4'$ in the frontier. Obviously, Υ'' is a valid derivation tree and Υ'' describes a derived tree whose frontier is still z. Thus, in $v_1'v_3'v_5'$, there are still $|z|$ occurences of substrings of the form β_a. Repeating the above argument, we will find a derivation tree Υ, given in Figure 4.9, which describes a tree γ with frontier z such that v_2v_4 has at least one occurence of subtree given in Figure 4.5. We can then conclude for every $i \geq 0$, the following derivation trees are valid derivation trees. Now consider the derivation tree (partial) and the tree it describes as shown in Figure 4.11. Since v_2v_4 contains at least one occurence of a subtree given in Figure 4.5, we can conclude that $w'w'' \in \Sigma^+$. Now

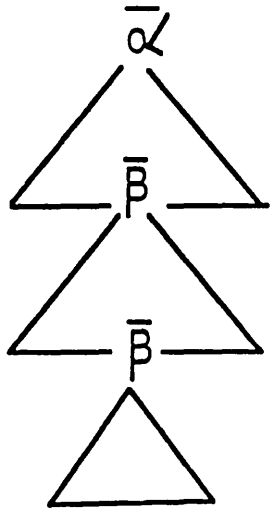

Figure 4.8: Derivation tree with recursion

consider the other two partial derivation trees in Figure 4.11 and the trees they describe. We can conclude that $w_1 w_2 \in \Sigma^*$ and $|z| \geq n$. According to Υ. Υ_1, the derived auxiliary tree $\gamma_1 \in \mathcal{D}(\beta)$ is adjoined in some node addressed j in γ_1. The resulting tree is adjoined in γ_1 at the same node addressed j. This process is repeated i times. There are three cases to consider – when node addressed j in γ_1 lies

1. on the spine (the path from the root to the foot)
2. to the left of the spine
3. to the right of the spine

Case 1: derivation of γ when node addressed j is on the spine of γ_1. γ_1 can be written as in figure 4.12. Considering the derivation tree from bottom-up, we will have γ_3 first adjoining in the node addressed j in γ_1 if $i \geq 1$.

This tree is adjoined into γ_1 at the node addressed j. the process is repeated i times. Now the derived tree obtained is adjoined in γ_3, the derived tree described by Υ_3. The final derived tree has a frontier given by $u_1 v_1^i w_1 v_2^i u_2 v_3^i w_2 v_4^i u_3$ We can therefore conclude that if $z \in L$ and $|z| \geq n$, then z can be written as $u_1 v_1 w_1 v_2 u_2 v_3 w_2 v_4 u_3$,

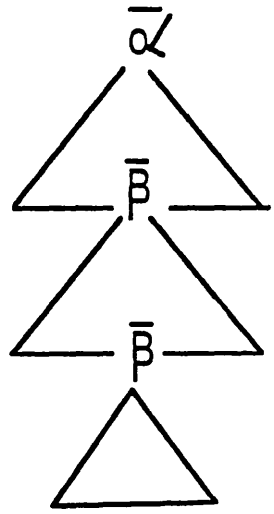

Figure 4.9: Derivation Tree in a recursive derivation

where $|v_1w_1v_2v_3w_2v_4| \leq n$ and $|v_1v_2v_3v_4| \geq 1$ and for all $i \geq 0$ it is true that $u_1v_1^iw_1v_2^iu_2v_3^iw_2v_4^iu_3 \in L$.

Case 2: The node addressed j is to the left of the spine of γ_1. Then γ_1 can be written as in Figure 4.13.

γ_2 and γ_3 remain the same as before. We can thus conclude that with the same conditions as that in the Case 1, we will have for $i \geq 0$, $u_1v_1^{i+1}w_1v_2w_2(v_3v_2v_4)^iv_3u_2v_4u_3 \in L$.

Case 3: The node addressed j is to the right of the spine of γ_1. Working out similar to Case 2, we can thus conclude that with the same conditions as that in the Case 1, we will have for $i \geq 0$, $u_1v_1u_2v_2(v_1v_3v_2)^iw_1v_3w_2v_4^{i+1}u_3 \in L$.

Note that the original string and the strings obtained after pumping in cases 2 and 3, may be rewritten such that they correspond to subcases of case 1. Thus, the pumping lemma may be stated as follows.

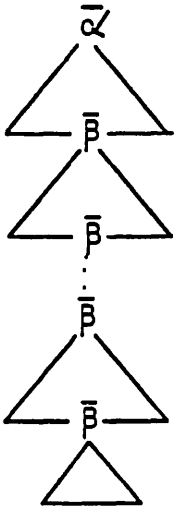

Figure 4.10: Derivation Tree with i-fold recursion

Theorem 4.7 If L is a TAL, then there is a constant n such that if $z \in L$ and $|z| \geq n$ then z may be written as $z = u_1v_1w_1v_2u_2v_3w_2v_4u_3$ with $|v_1w_1v_2v_3w_2v_4| \leq n$, $|v_1v_2v_3v_4| \geq 1$ such that for all $i \geq 0$, $u_1v_1^iw_1v_2^iu_2v_3^iw_2v_4^iu_3 \in L$.

With this pumping lemma we can show that the language $\{a^nb^nc^nd^ne^n \mid 0 \leq n\}$ is not a TAL. If k is the constant of the pumping lemma, then consider the string $a^kb^kc^kd^ke^k$. Since we have $|v_1w_1v_2v_3w_3v_4| \leq k$, this substring can have at most four different symbols, and has length of at least one. Thus pumping these substrings will result in the occurence of at least one symbol remaining the same while increasing the number of occurences of some other symbol. Thus, the string we will obtain will not have equal number of $a's, b's, c's, d's, e's$. Thus, this language is not a TAL.

4.3 Semi-Linearity of TAL's

In this section we show that every TAL is a semilinear language. The property of semi-linearity (and a closely related property called the *constant growth property*) of languages has been discussed in the context of natural languages in literature (see Joshi [Joshi 85]

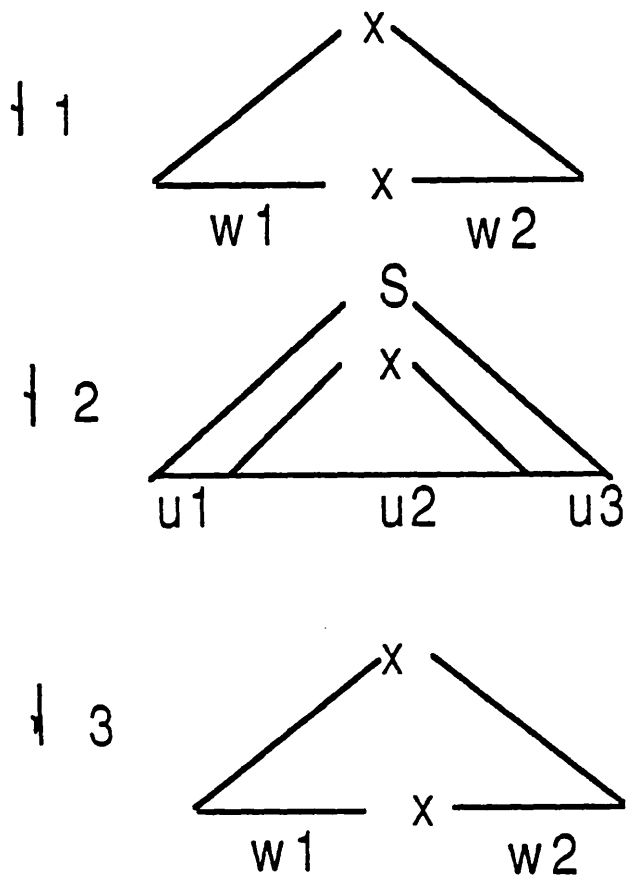

Figure 4.11: Trees described by partial derivation trees

and Berwick and Weinberg [Berwick 84]). The property of semilinearity is concerned about the occurence of the different alphabet symbols in strings of a language and whether this count is ultimately periodic. Hence, it suggests that as long as only the alphabet count is concerned, a semilinear language can not be distinguished from Regular Sets.

Parikh [Parikh 66] first discussed the semilinearity of languages. He showed that any CFL is a semilinear language. Later [Ginsburg 68] studied AFL's which contain only semilinear languages and what properties follow from it.

We shall first define the semilinear property

Definition 4.1 Semilinear Sets

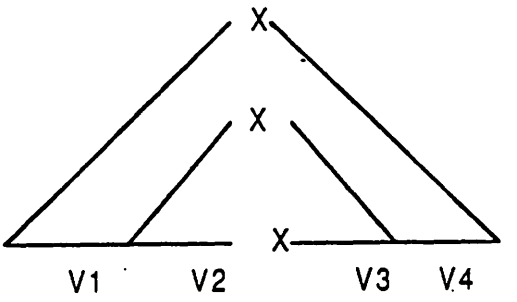

Figure 4.12: Case 1: j is on the spine

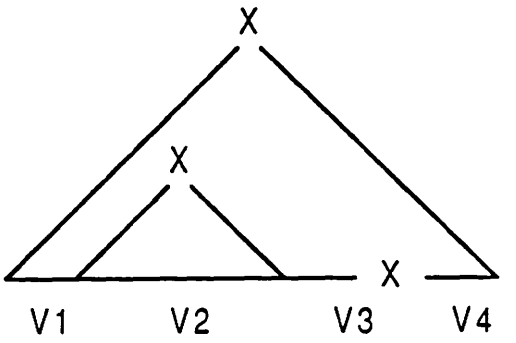

Figure 4.13: j to the left of the spine

Let B and P be finite sets over $\mathcal{N}^n$. We define

$$L(B.P) = \{b + p_1 + \cdots + p_k : b \in B.k \geq 0.p_1, \ldots p_k \in P\}$$

Any set $L(P.B)$, where B and P are finite subsets of $\mathcal{N}^n$, is said to be **linear**. A **semilinear** set is a finite union of linear sets.

Definition 4.2 Parikh Mapping

The Parikh mapping Ψ maps strings over Σ^* (where $\Sigma = \{a_1, \ldots, a_n\}$) to n-tuples over natural numbers, $\mathcal{N}^n$, and is defined recursively as follows

$$\Psi(a_i) = (p_1, \ldots, p_i, \ldots, p_n)$$

where $p_j = 1$ if j=i or 0 otherwise.

The Parikh mapping, Ψ, can then be extended to mappings for a string in the following way.

$$\Psi(w_1 \ldots w_m) = \Psi(w_1) + \cdots + \Psi(w_m)$$

where $w_1, \ldots, w_m \in \Sigma^*$

The definition of Ψ can be extended to languages as follows
$\Psi(L) = \Sigma_{w \in L} \Psi(w)$

Definition 4.3 Semilinear Languages

A language L is said to be a semilinear language if $\Psi(L)$ is a semilinear set. From the definition of semilinearity, it follows that every semilinear language is letter equivalent to a regular language, i.e., if L is a semilinear language then there exists a regular language, R, such that $\Psi(L) = \Psi(R)$.

Definition 4.4 Constant Growth Property

The length of a string, denoted by $| w |$ is a mapping from Σ^* to $\mathcal{N}$. We can extend this mapping to mappings for sets of strings in the obvious manner. A language is said to have constant growth property if this mapping of strings of this language to a subset of $\mathcal{N}$ (denoting lengths of strings in the language) yields a semilinear set.

Semilinearity and constant growth property (a consequence of semilinearity) has received attention in the context of natural languages (for example, Joshi [Joshi 85] and Berwick and Weinberg [Berwick 84]). As noted by [Joshi 85,Berwick 84] although these properties are not structural properties, they depend on the structural property that sentences can be built from a finite set of clauses of bounded structure. In natural languages, it seems reasonable to assume that there are no sudden jumps in the length of sentences and that the set of lengths of the sentences is ultimately periodic. The property that

we would like to capture, however, is the linear growth in structure. Semilinearity and constant-growth properties are only approximations to this property we wish to capture.

Parikh [Parikh 66] had shown that every CFL is a semilinear language. Thus, in order to show that a language is a semilinear language, it is sufficient to show that it is letter equivalent to a CFL. Since we know that every TAL is also a MHL, we can show that every TAL is a semilinear language if we show that every MHL is semilinear.

Theorem 4.8 For every MHG, $G = (N, \Sigma, S, P)$, there is a CFG, $G' = (N, \Sigma, S, P')$, such that $\Psi(L(G)) = \Psi(L(G'))$.

We assume that every production in P has the form

$$A \rightarrow f(B, C) \quad f \in \{W, C1, C2\}$$

$$A \rightarrow \overline{w} \qquad w \in \Sigma^*$$

P' is defined as follows:

$$A \rightarrow f(B, C) \in P \qquad iff \qquad A \rightarrow BC \in P'$$

$$A \rightarrow \overline{w} \in P \qquad iff \qquad A \rightarrow w \in P'$$

Lemma 4.1 $\Psi(L(G)) = \Psi(L(G'))$

We show by induction that $A \overset{k}{\underset{G}{\Longrightarrow}} w_1 w_2$ if and only if $A \overset{s}{\underset{G'}{\Longrightarrow}} w$ such that $\Psi(w_1 w_2) = \Psi(w)$.

The base case corresponds to the productions $A \rightarrow \overline{w} \in P$ and iff $A \rightarrow w \in P'$ and is true by definition.

Inductive step: Let us assume that $B \overset{i}{\underset{G}{\Longrightarrow}} u_1 u_2$ and $C \overset{v}{\underset{G}{\Longrightarrow}} v_1 v_2$, where $i, j \leq k$. By the inductive hypothesis this is true if and only if the following is true. $B \overset{i}{\underset{G'}{\Longrightarrow}} u$ and $C \overset{j}{\underset{G'}{\Longrightarrow}} v$ where $\Psi(u_1 u_2) = \Psi(u)$ and $\Psi(v_1 v_2) = \Psi(v)$. Let $A \rightarrow f(B, C) \in P$ and hence $A \rightarrow BC \in P'$. Regardless of the function f used in the production, $A \overset{i+j}{\underset{G'}{\Longrightarrow}} w_1 w_2$

where $\Psi(w_1w_2) = \Psi(u_1u_2) + \Psi(v_1v_2)$; and $A \overset{i+j-1}{\underset{G'}{\Longrightarrow}} w$ where $\Psi(w) = \Psi(u) + \Psi(v) = \Psi(u_1u_2) + \Psi(v_1v_2)$.

4.4 Parsing Algorithm for TAL's

The algorithm, we present here to parse Tree Adjoining Languages (TALs), is a modification of the CYK algorithm (which is described in detail in [Aho 73b]), which uses a dynamic programming technique to parse CFL's. For the sake of making our description of the parsing algorithm simpler, we shall present the algorithm for parsing without considering local constraints. We will later show how to handle local constraints.

We shall assume that any node in the elementary trees in the grammar has at most two children. This assumption can be made without any loss of generality, because it can be easily shown that for any TAG G there is an equivalent TAG G_1 such that any node in any elementary tree in G_1 has atmost two children. A similar assumption is made in CYK algorithm. We also assume that the frontier of each auxiliary tree has at least one terminal symbol. We use the terms ancestor and descendant, throughout the paper as a transitive and reflexive relation: for example, the foot node may be called the ancestor of the foot node.

The algorithm works as follows. Let $a_1 \ldots a_n$ be the input to be parsed. The algorithm is an extension of the CKY algorithm and is based on the *dynamic programming* technique. In the CKY algorithm to parse CFG's, a two-dimensional array, say A, storing subsets of nonterminals. A is used to keep track of the boundaries in the input substring which are derived from nonterminals. Thus if a nonterminal, X, of the grammar derives the substring $a_i \ldots a_j$ of the input $a_1 \ldots a_n$, then X is stored in the element $A[i,j]$ of the input. Since concatenation is the only operation that is used in CFG's, given a rule $Z \rightarrow XY$ in a CFG, G, we let Z be stored in $A[i,j]$ if $X \in A[i,k]$ and $Y \in A[k+1,j]$ for some k, since $Z \stackrel{*}{\underset{G}{\Rightarrow}} a_i a_j$ if it is the case that $X \stackrel{*}{\underset{G}{\Rightarrow}} a_i a_k$ and $Y \stackrel{*}{\underset{G}{\Rightarrow}} a_{k+1} a_j$. We extend this algorithm to parse TAG's. We chose to use a four-dimensional array for the following reason. Unlike the concatenation operation in CFG's, adjunction does not operate on contiguous substrings of the input. Suppose a node in some tree dominates

a frontier which has the substring $a_i a_j$ to the left of the foot node of this tree and the substring $a_k a_l$, on the right of the foot node. These substrings need not be a continous part of the input; in fact; when this tree is used for adjunction then a string is inserted between these two substrings. Thus, in order to represent a node, we characterize it by two strings that appear to the left and right of the foot node, and is in the case of the CKY algorithm we need two indices for each substring. Thus, in our algorithm, we use a four-dimensional array A; each element of the array contains a subset of the names of nodes of elementary trees. We say a node, addressed η, of an elementary tree γ belongs to A[i,j,k,l] if it satisfies the following proposition

Proposition 4.1

The node given by $\langle \gamma, \eta \rangle \in A[i,j,k,l]$ if and only if there is a tree $\gamma' \in \mathcal{D}(\gamma, \eta)$ whose frontier is given by either $a_{i+1}...a_j Y a_{k+1}...a_n$ (where the foot node of γ is labelled by Y) or $a_{i+1}...a_n$ (i.e., $j = k$). This corresponds to the case when γ is a sentential tree, or $\langle \gamma, \eta \rangle$ does not dominate the foot node). The indices (i,j,k,l) refer to the positions between the input symbols rather than refer to input symbols, and hence range over 0 through n. If i = 5 say, then it refers to the gap between a_5 and a_6.

We initially fill A[i,i+1,i+1,i+1] with those nodes in the frontier of the elementary trees whose label is the same as the input a_{i+1} for $0 \leq i \leq n\text{-}1$. The foot nodes of auxiliary trees will belong to all A[i,i,j,j], such that $i \leq j$.

We are now in a position to fill in all the elements of the array A. There are five cases to be considered.

Case 1. We know that if a node μ_1 in a derived tree is the ancestor of the foot node, and node μ_2 is its right sibling, such that $\mu_1 \in$ A[i,j,k,l] and $\mu_2 \in$ A[l,m,m,n], then their parent, say, μ should belong to A[i,j,k,n], see Figure 4.14.

Case 2. If the right sibling μ_2 is the ancestor of the foot node such that it belongs to A[l,m,n,p] and its left sibling μ_1 belongs to A[i,j,j,l], then we know that the parent μ of μ_1 and μ_2 belongs to A[i,m,n,p], see Figure 4.15.

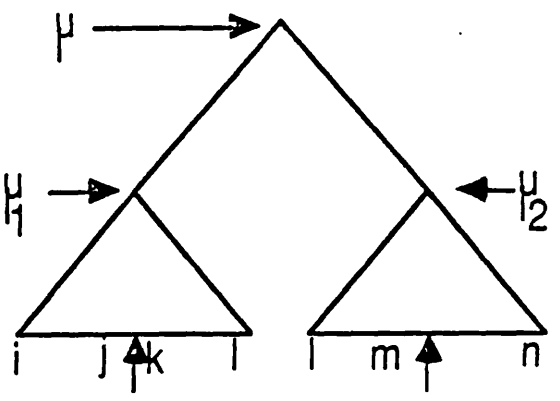

Figure 4.14: Parsing algorithm case 1

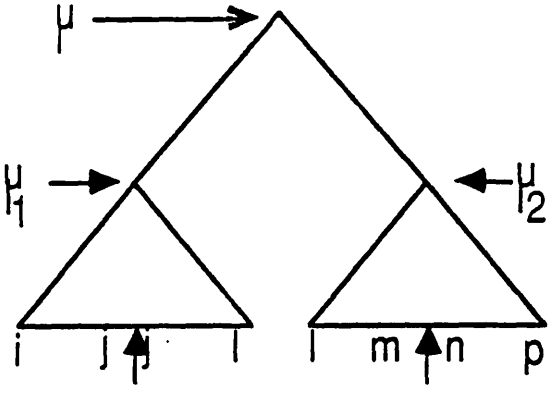

Figure 4.15: Parsing algorithm case 2

Case 3. If neither μ_1 nor its right sibling μ_2 are the ancestors of the foot node (or there is no foot node) then if $\mu_1 \in A[i.j.j.l]$ and $\mu_2 \in A[l.m.m.n]$ then their parent μ belongs to A[i,j,j,n].

Case 4. If a node μ has only one child μ_1, and if $\mu_1 \in A[i.j.k.l]$, then obviously $\mu \in$ A[i,j,k,l].

Case 5. If a node $\mu_1 \in$ A[i,j,k,l], and the root μ_2 of a derived tree γ having the same label as that of μ_1, belongs to A[m,i,l,n], then adjoining γ at μ_1 makes the resulting node to be in A[m,j,k,n], see Figure 4.16.

Since the elements of the array contain a subset of the addresses of nodes in the elementary

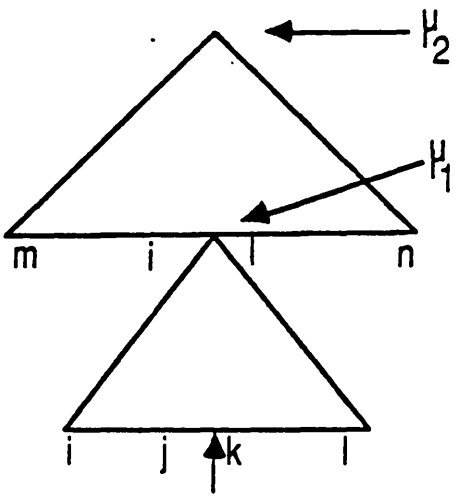

Figure 4.16: Parsing algorithm case 5

trees, the the siμe of any set is bounded by a constant, determined by the grammar. The complete algorithm is given in [VijayShanker 85]. It has been shown in that paper that the complexity of the algorithm is $O(n^6)$ where n is the length of the input.

4.4.1 The Parsing Algorithm

The complete algorithm is given below

Step 1 For i=0 to n-1 step 1 do

Step 2 put all nodes in the frontier of elementary trees whose label is $a_i - 1$ in A[i,i+1,i+1,i+1].

Step 3 For i=0 to n-1 step 1 do

Step 4 for j=i to n-1 step 1 do

Step 5 put foo nodes of all auxiliary trees in A[i,i,j,j]

Step 6 For l=0 to n step 1 do

Step 7 For i=l to 0 step -1 do

Step 8 For j=i to l step 1 do

Step 9 For k=l to j step -1 do

Step 10 do Case 1

Step 11 do Case 2

Step 12 do Case 3

Step 13 do Case 5

Step 14 do Case 4

Step 15 Accept if root of some initial tree $\in A[0,j,j,n]$, $0 \leq j \leq n$

where,

(a) Case 1 corresponds to situation where the left sibling is the ancestor of the foot node. The parent is put in A[i,j,k,l] if the left sibling is in A[i,j,k,m] and the right sibling is in A[m,p,p,l], where $k \leq m < l$, $m \leq p$, $p \leq l$. Therefore Case 1 is written as

for m=k to l-1 step 1 do

for p= m to l step 1 do

if there is a left sibling in A[i,j,k,m] and the right sibling in A[m,p,p,l] satisfying appropriate restrictions then put their parent in A[i,j,k,l].

(b) Case 2 corresponds to the case where the right sibling is the ancestor of the foot node. If the left sibling is in A[i,m,m,p] and the right sibling is in A[p,j,k,l], $i \leq m < p$ and $p \leq j$, then we put their parent in A[i,j,k,l]. This may be written as

for m=i to j-1 step 1 do

for p=m+1 to j step 1 do

for all left siblings in A[i,m,m,p] and right siblings in A[p,j,k,l] satisfying appropriate restrictions put their parents in A[i,j,k,l].

(c) Case 3 corresponds to the case where neither children are ancestors of the foot node. If the left sibling $\in$ A[i,j,j,m] and the right sibling $\in$ A[m,p,p,l] then we can put the parent in A[i,j,j,l] if it is the case that ($i < j \leq m$ or $i \leq j < m$) and ($m < p \leq l$ or m

$\leq p < l$). This may be written as

for m = j to l-1 step 1 do

for p = j to l step 1 do

for all left siblings in A[i,j,j,m] and right siblings in A[m,p,p,l] satisfying the appropriate restrictions put their parent in A[i,j,j,l].

(e) Case 5 corresponds to adjoining. If μ_1 is a node in A[m,j,k,p] and μ_2 is the root of a auxiliary tree with same symbol as that of μ_1, such that μ_2 is in A[i,m,p,l] ($(i \leq m \leq p < l$ or $i < m \leq p \leq l)$ and $(m < j \leq k \leq p$ or $m \leq j \leq k < p)$). This may be written as

for m = i to j step 1 do

for p = m to l step 1 do

if a node $\mu_1 \in$ A[m,j,k,p] and the root of auxiliary tree is in A[i,m,p,l] then put μ_1 in A[i,j,k,l]

(d) Case 4 corresponds to the case where a node μ_2 has only one child μ_1

If $\mu_1 \in$ A[i,j,k,l] then put μ_2 in A[i,j,k,l]. Repeat Case 4 again if μ_2 has no siblings.

4.4.2 Parsing with Local Constraints

So far,we have assumed that the given grammar has no local constraints. If the grammar has local constraints, it is easy to modify the above algorithm to take care of them. Note that in Case 5, if an adjunction occurs at a node μ_1, we add μ_1 again to the element of the array we are computing. This seems to be in contrast with our definition of how to associate local constraints with the nodes in a sentential tree. We should have added the root of the auxiliary tree instead to the element of the array being computed, since so far as the local constraints are concerned, this node decides the local constraints at this node in the derived tree. However, this scheme cannot be adopted in our algorithm for obvious reasons. Therefore, we let pairs of the form (X,C) belong to elements of the

array, where μ_1 is as before and C represents the local constraints to be associated with this node.

We then alter the algorithm as follows. If (X,C_1) refers to a node at which we attempt to adjoin with an auxiliary tree (whose root is denoted by (Y,C_2)), then adjunction would determined by C_1. If adjunction is allowed, then we can add (X,C_2) in the corresponding element of the array. In cases 1 through 4, we do not attempt to add a new element if any one of the children has an obligatory constraint.

The proof of correctness of this algorithm is given in [VijayShanker 85]. It has been shown that the algorithm works in $O(n^6)$, where n is the length of the input. Note, however, that this algorithm works only for the Tree Adjoining Grammars whose auxiliary trees have at least one terminal symbol in the frontier. In order to take care of the general case, we can find an equivalent TAG such that every derived auxiliary tree (without OA constraints) has at least one terminal symbol in the frontier. One procedure to find such a TAG works as follows. Given a TAG, we find a MHG for it. We can then remove nonterminals from this grammar which derive empty split strings. We can then convert this MHG back into a TAG (similar to the procedure used in showing the equivalence of TAG's and MHG's) such that every derived auxiliary tree without OA constraints has at least one terminal in the frontier. Once such a TAG is found, with minor modifications to the above algorithm, we can prove the algorithm works correctly for any TAL.

4.4.3 Retrieving a Parse

Once it has been determined that the given string belongs to the language, we can find the parse in a way similar to the scheme adopted in CYK algorithm.To make this process simpler and more efficient, we can use pointers from the new element added to the elements which caused it to be put there. For example, consider Case 1 of the algorithm

(step 10). If we add a node μ to A[i,j,k,l], because of the presence of its children μ_1 and μ_2 in A[i,j,k,m] and A[m,p,p,l] respectively, then we add pointers from this node μ in A[i,j,k,l] to the nodes μ_1s, μ_2 in A[i,j,k,m] and A[m,p,p,l]. Once this has been done, the parse can be found by traversing the tree formed by these pointers.

Palis, Shende, and Wei [Palis 86] have also given a way of retreiving the parse from this array.

Chapter 5

Feature Structure Based Tree Adjoining Grammars

Several different approaches to natural language grammars have developed the notion of *feature structures* to describe linguistic objects. These include lingusitic theories such as Generalized Phrase Structure Grammars (GPSG) [Gazdar 85b], Lexical Functional Grammars (LFG) [Kaplan 83] and formalisms such as Functional Unification Grammars (FUG) [Kay 79] and PATR-II [Shieber 85] which were developed as computational tools.

A feature structure is essentially a set of attribute-value pairs where values may be atomic symbols or another feature structure. Feature structures can also be characterized as either constants (atomic values) or partial functions from labels to feature structures (reference [Pereira 84]). The name "unification-based grammatical formalisms" or "unificational grammars" have been used in literature because *unification* of the feature structures is the primary operation used in combining feature structures.

5.1 Unification Grammars

In order to capture certain linguistic phenomena such as agreement, subcategorization, etc., a number of recent grammatical systems have added, on top of a CFG skeleton, a feature based informational element. Example of such systems include Generalized Phrase Structure Grammars (GPSG), Lexical functional Grammars (LFG), and Head-driven Phrase Structure Grammars (HPSG). Although the notation for these informational elements may differ between formalisms, in all the above-mentioned systems, the informational elements may be thought of as feature structures with unification as the main operation on these feature structures. In the rest of our work, we will consider PATR-II as a representative of the unification-based grammatical systems which are based on a CFG-skeleton. PATR-II was developed as a language to express the various linguistic theories found in these unification based approaches, and is the least common denominator of these unification systems.

5.1.1 Feature Structures and the PATR-II system

In the PATR-II system, equations are expressed along with the productions of an underlying context free grammar. For example,

$$S \rightarrow X_1 X_2$$

$$X_0.cat = S$$
$$X_1.cat = NP$$
$$X_2.cat = VP$$
$$X_1.agr = X_2.agr$$
$$X_2.subject = X_1$$

states that S rewrites to NP and VP, with the agreement feature of the NP being the same as that of the VP and that the subject of the VP is the NP. This statement is represented by the following feature structure.

$$\begin{bmatrix} cat: & S & \\ 1: & \boxed{b}\begin{bmatrix} cat: & NP \\ agr: & \boxed{a} \end{bmatrix} & \\ 2: & \begin{bmatrix} cat: & VP \\ agr: & \boxed{a} \\ subject: & \boxed{b} \end{bmatrix} & \end{bmatrix}$$

The notation of the co-indexing box ($\boxed{a}$ and $\boxed{b}$ in this example) is used to express the fact that the values of two subfeatures are the same. For example, the agr values of X_1 and X_2 are co-indexed by $\boxed{a}$ to state that their agreement values are the same; and $\boxed{b}$ is used to state that the subject of X_2 is the the entire feature structure corresponding to X_1. Feature structures with co-indexing boxes have also been called *reentrant* feature structures in the literature. Co-indexing boxes are used to state that two or more feature structures have the *same* value as distinguished from the weaker notion of *similar* values(see [Shieber 85]). As will be clear from the discussion later, the two feature structures (example taken from [Shieber 85]) below have to be distinguished.

$$\begin{bmatrix} f: & [h:a] \\ g: & [h:a] \end{bmatrix} \text{ and } \begin{bmatrix} f: & \boxed{1}[h:a] \\ g: & \boxed{1} \end{bmatrix}$$

For example, on unifying with the feature structure, $[g : [l : b]]$, results in following two different feature structures.

$$\begin{bmatrix} f: & [h:a] & \\ g: & \begin{bmatrix} h: & a \\ l: & b \end{bmatrix} & \end{bmatrix} \text{ and } \begin{bmatrix} f: & \boxed{1}\begin{bmatrix} h: & a \\ l: & b \end{bmatrix} \\ g: & \boxed{1} & \end{bmatrix}$$

We can define a partial ordering, $\sqsubseteq$, on a set of feature structures using the notion of *subsumption* (carries *less information* or is *more general*). Using the partial function representation of feature structures, we say

1. $[a] \sqsubseteq [a]$ for each atomic value a
2. $F_1 \sqsubseteq F_2$ if the partial function F_2 is defined for all the labels for which F_1 is defined and furthermore for each label l such that $F_1(l)$ is defined, $F_1(l) \sqsubseteq F_2(l)$.

We use [ ] to represent the totally unspecified feature structure. Then, $[\,] \sqsubseteq F$ for all feature structures F. Unification of two feature structures (if it is defined) corresponds to the least feature structure that has all the information contained in the original two feature structures. Thus, unifiying two feature structures gives rise to the feature structure that is the least upper bound of the original two feature structures (i.e., l.u.b according to the subsumption relation). Note, however, that the unification of two feature structures may be undefined. Two feature structures which have different values for the same attribute cannot be unified. For example,

$$\begin{bmatrix} l_1 : & a \\ l_2 : & \begin{bmatrix} l_3 : & b \\ l_4 : & c \end{bmatrix} \end{bmatrix} \text{ and } \begin{bmatrix} l_1 : & a \\ l_2 : & \begin{bmatrix} l_5 : & e \\ l_4 : & d \end{bmatrix} \end{bmatrix}$$

cannot be unified because the feature structures that are the values of the l_2 attribute cannot be unified (since the values for the attribute l_4 are c and d).

The unification algorithm works as follows. The feature structure, f, obtained by unifying the feaure structures f_1 and f_2 is defined as follows:

- If f_1 and f_2 are both defined for the attribute l, then the value for the attribute l in f is the unification of the values for the attribute l in f_1 and f_2.

- If one of the feature structures, f_1 and f_2, is not defined for an attribute l, then the value for the attribute l in f is the the value of the attribute l in the feature structure which is defined for this attribute.
- An atom, a, does not unify with a complex feature structure or the atom b, where $a \neq b$.
- if $f_1 = f_2$ then $f = f_1 = f_2$.

For more details on the feature structures and unification, see [Shieber 85].

Feature structures can be encoded as a directed acyclic graph (DAG), where the edges are labeled by the labels in the feature structure. Shared values (represented in feature structures by co-indexing boxes) are represented by the joining of arcs. We shall using the two representations interchangeably.

Derivations in PATR-II are similar to those in CFG's except for the derivations of feature structures. If X_1 and X_2 derive strings w_1 and w_2 and feature structures F_1 and F_2 respectively, then given a rule $X_0 \rightarrow X_1 X_2$ with an associated set of equations E, then if F_0 is the unifier of F_1 and F_2 according to the equations in E then X_0 derives the string $w_1 w_2$ and the feature structure F_0. On the other hand, if unification fails then the rule is not applicable.

We will now show the derivation of the sentence *John sleeps* assuming the association of feature structures with the lexicon given below. We also show why the string *John sleep* is not generated by the grammar, whose productions are given by

S → NP VP (1)

S.subject = NP

S.pred = NP

NP.agr = VP.agr

VP.subject = NP

NP → N (2)

NP.agr = N.agr

VP → V (3)

VP.agr = V.agr

John – [agr:[number:singular]]

sleeps – [agr:[number:singular]]

sleep – [agr:[number:plural]]

For the derivation of the sentence *John sleeps*, we derive the following feature structure

$$\begin{bmatrix} subject: & [agr:[number:singular]\boxed{a}]\boxed{b} \\ pred: & \begin{bmatrix} agr: & \boxed{a} \\ subject: & \boxed{b} \end{bmatrix} \end{bmatrix}$$

whereas, the sentence *John sleep* is not derivable because the following two feature structures are not unifiable according the equations associated with the production (1).

$$[agr:[number:singular]] \qquad \begin{bmatrix} agr: & [number:plural] \\ subject: & [\,] \end{bmatrix}$$

5.1.2 Logical Representation

So far, we have given an informal description of feature structures and the unification of feature structures. Pereira and Sh eber [Pereira 84] have given a semantics of feature structures using Scott's domain theory [Schmidt 87] and modeling feature structures as partial functions. Lauri Kartunnen [Kartunen 85] has argued for the need of disjunctive

specification, which is used extensively in FUG and Systemic Grammars. To model feature structures with disjunction, would involve the power set construction, as observed by Pereira and Sheiber [Pereira 84]. A simpler definition can be made using the logical formulation of feature structures given by Rounds and Kasper [Rounds 86,Kasper 86]. Feature structures in this system are specified as formulae in a version of modal propositional logic (henceforth referred as Rounds-Kasper logic). These formulae can be reduced to a normal form using logical equivalences given in [Rounds 86,Kasper 86]. Extending on the work of Ait-Kaci [AitKaci 85], they show that deterministic finite state automata (whose transition graph is a DAG) can be used to form a logical model and the satisfiability of the formulae can be given in terms of these automata.

We now describe the logic of Rounds and Kasper briefly. The details may be found in [Rounds 86,Kasper 86]

Rounds-Kasper Logic

The work of Rounds and Kasper [Rounds 86,Kasper 86] was motivated to give a complete algebraic account of the logical properties of feature structures, which can be used for computational manipulation and mathematical analysis. Their model was developed to give a precise account of feature structures that also include disjunctive specification. The need for adding disjunctive specification to feature structures have been discussed by Kartunnen [Karttunen 85]; disjunctive specification have also been used in FUG [Kay 79].

Rounds-Kasper logic is a version of propositional modal logic. In addition to conjunction, disjunction, the logic uses a modal operator ":" and allows equality statements between paths. A formula $l : \phi$ is a logical representation of a feature structure whose value for attribute l is a feature structure satisfying the formula ϕ.

Given the set of atoms, C, and the set of labels, L, the following are the well formed

formulae in the Rounds-Kasper logic:

NIL

TOP

a

$l : \phi$

$\phi \wedge \psi$

$\phi \vee \psi$

$\{p_1, \ldots, p_n\}$

where $A \in C$, $l \in L$, ϕ, ψ are well-formed formulae. NIL and TOP convey "no information" and "inconsistent information" respectively. Each p_i represents a path of the form $l_{i,1} : l_{i,2} : \ldots : l_{i,n_i}$ respectively. This formula is interpreted as $p_1 = \ldots = p_n$. For example, the formula $f : g : a \wedge \{h : j, f : g\}$ represents the feature structure below. The formulae in this logic are only notational variant of the feature structure representation.

$$\begin{bmatrix} f: & [g : \boxed{1}[a]] \\ h: & [j : \boxed{1}] \end{bmatrix}$$

As suggested in [AitKaci 85], deterministic finite state automata (dfa) model the feature structures. Satisfiability of the formulae may be defined in terms of these finite state automata.

As noted in [Rounds 86], deterministic finite state machines have the following desirable properties as domain for feature structures (see [Kasper 86,Rounds 86])

1. The value of any defined path can be denoted by the state of the automaton
2. Finding the value of a path is interpreted by running the automaton on the path string
3. The DFAs captures the crucial properties of shared structure

(a) two paths which are unified have the same state as a value

(b) unifiaction is equivalent to a state-merge operation

Let $\mathcal{A} = (Q, L, C, \delta, \lambda, q_0, F)$ be a dfa whose transition graph is a directed acyclic graph, where L is a set of labels, C is a set of atoms, and λ is an output function with the output alphabet C. $\mathcal{A}$ satisfies a formula ϕ ($\mathcal{A} \models \phi$) if the following is the case.

$\phi = NIL$		
$\phi = a$	iff	$\lambda(q_0) = a$
$\phi = l : \phi_1$	iff	$\mathcal{A}/l \models \phi_1$
$\phi = \phi_1 \wedge \phi_2$	iff	$\mathcal{A} \models \phi_1$ and $\mathcal{A} \models \phi_2$
$\phi = \phi_1 \vee \phi_2$	iff	$\mathcal{A} \models \phi_1$ or $\mathcal{A} \models \phi_2$
$\mathcal{A} \models \{p_1, \ldots, p_n\}$	iff	$p_1, \ldots, p_n$ are in the same Nerode-equivalence class of $\mathcal{A}$.

$\mathcal{A}/l$ is the automaton obtained by making the state $\delta(q_0, l)$ as the initial state. p_1, p_2 are in the same Nerode-equivalence class if $\delta(q_0, p_1) = \delta(q_0, p_2)$. No automaton satisfies the formula TOP since TOP conveys inconsistent information.

[Rounds 86,Kasper 86] also give a list of logical equivalences which may be used to reduce any formula to a normal form. If the formula is not satisfiable then it reduces to TOP. We now list the algebraic laws for manipulating the logical formulae which have been given in [Rounds 86,Kasper 86].

Failure

$$l : TOP = TOP \qquad (1)$$

Conjunction (unification)

$$\phi \wedge TOP = TOP \qquad (2)$$

$\phi \wedge NIL = \phi$ (3)

$a \wedge b = TOP, \forall a, b \in A$ and $a \neq b$ (4)

$a \wedge l : \phi = TOP$ (5)

$l : \phi \wedge l : \psi = l : (\phi \wedge \psi)$ (6)

Disjunction

$\phi \vee NIL = NIL$ (7)

$\phi \vee TOP = \phi$ (8)

$l : \phi \vee l : \psi = l : (\phi . \vee \psi)$ (9)

Path Equivalence

$E_1 \wedge E_2 = E_2$ whenever $E_1 \subseteq E_2$ (10)

$E_1 \wedge E_2 = E_1 \wedge (E_2 \cup \{zy \mid z \in E_1\})$ (11)

for any y such that $\exists x : x \in E_1$ and $xy \in E_2$

$E \wedge x : c = E \wedge (\wedge_{y \in E} y : c)$ where $x \in E$ (12)

$E = E \wedge \{x\}$ if x is a prefix of a string in E (13)

$l : E = \{lw \mid w \in E\}$ (14)

$\{\epsilon\} = NIL$ (15)

$E = TOP$ for any E such that there are strings (16)

$x, xy \in E$ and $y \neq \epsilon$

In addition to laws (1)-(16), there are laws for commutativity of *and* and *or*, distributivity, associativity, idempotence, and absorption. The consistency and completeness of these equivalences has also been shown in [Rounds 86]. [Rounds 86,Kasper 86] also show that the consistency problem for these formulae is an NP-complete problem.

5.1.3 Unification Systems and Domain of Locality

Most of the recent unification-based grammatical formalisms can be viewed as notational variant of CFG-based unificational approaches. These systems are very powerful and can express most of the theories for natural language syntax. They have the advantage of being declarative, with facility to state the linguistic phenomena directly. Equality testing, pattern matching and feature passing are found in a wide range of linguistic analyses – in concurrence with the reliance of unification in these systems. The interpretation of the formalism is order independent because of the associativity of its operations.

Although the generality of these unificational systems is attractive, they suffer from being computationally inefficient. Some of this inefficiency arises out of the limited "domain of locality" in CFG's[1]. In the CFG based unificational systems, equations are associated with individual rewrite rules. Thus, in order to check constraints between different constituents, feature structures have to be passed around extensively. Features of the "dependent" items have to be passed to their parents. In fact, they have to be passed upto the common ancestor, since the constraints are stated with respect to individual productions. For example, to check that the agreement features of the head noun and the main verb, the agreement features are passed upto the topmost S node (or the node immediately dominating the subject NP and VP nodes. The extent of feature passing is decided by how far apart they are in the derivation tree. The feature passing mechanism is one of the main causes of the inefficiencies of the unification based systems. Some grammatical systems, e.g., GPSG, try to restrict the size of the feature structures that are unified and passed around in order to restrict the power of the systems.

Tree Adjoining Grammars is a grammatical formalism which provides a larger domain of locality. The theory underlying grammars for natural languages expressed in TAG formalism assumes that the dependencies between lexical items are encapsulated within

[1] We use the term "domain of locality" to denote the domain over which equations expressing constraints may be specified.

the elementary trees. This assumption may be summarized as

- Verbs and their complements are localized and are part of the same elementary tree.
- Dependent items belong to the same elementary tree or are in an elementary tree and the *immediate* tree adjoined into it.

Our major goal in embedding TAG's in an unificational framework is to capture this localization of dependencies. This is important for two reasons. First it is of linguistic interest to see the extent to which localization can be carried out. Secondly, it is of significance from the point of view of efficient computation due to constrained feature passing. Therefore, we would like to associate feature structures with the elementary trees themselves (rather than break these trees into a CFG-like rule based systems, and then use some mechanism to ensure only the trees produced by the TAG itself are generated[2]). In the feature structures associated with the elementary trees, we can state the constraints among the dependent nodes directly. Hence, in an initial sentence corresponding to a simple sentence, we can state that the main verb and the subject NP (which are part of the same initial tree) share the agreement feature. Thus, in principle, we do not have to pass the agreement features of the verb to its parent, the VP node, and then check the agreement feature of the siblings, the NP and VP nodes. No matter what tree is derived from this initial tree, the agreement of the verb and subject remains. Thus, this checking can be precompiled (of course only after lexical insertion) and need not be done dynamically as is the case in CFG-based unificational grammars. A similar situation arises in the case of subcategorization. In linguistic theory underlying TAG's, every element which is subcategorized by a verb, is a part of the same elementary tree that the verb belongs to. Thus, in the feature structures associated with this elementary tree,

[2]Such a scheme would be an alternate way of embedding TAG's in an unificational framework. However, it does not capture the linguistic intuitions underlying TAG's, and loses the attractive feature of localizing dependencies.

constraints regarding the subcategorization can be directly stated. Thus, we do not have to simulate a stack to capture the subcategorization information. Note, that the examples used above only suggest the advantages of associating feature structures with elementary trees as the building blocks of the formalisms. Considerable research in formalizing the linguistic theory for this system needs to be done before any formal investigation of the relative efficiency of processing can be done.

5.2 TAG's in an Unificational Framework

In this section, we will describe a method of embedding TAG's in an unificational framework. We will call this formalism *Feature structure based Tree Adjoining Grammars* or FTAG in short.

In an unification based grammar, equations stating some constraints among the elementary structures manipulated by a rule of the grammar are associated with the rewrite rule. Thus, these rules give the "domain of locality" to state the constraints. In TAG's, we wish to state the constraints among the nodes in the elementary trees, so that we can utilize the larger domain of locality of a TAG. In doing so, we hope, to the extent possible, minimize the flow of information and the number of linguistic stipulations that have to be made due to the limited domain of locality.

5.2.1 Domain of Locality in TAG's

As stated earlier, the linguistic theory underlying TAG's assumes that the elementary trees in a TAG give minimal linguistic structures and that the dependencies are captured locally. For example, even the so-called long distance dependencies are captured within a single elementary tree; the gap and the filler are a part of the same elementary tree,

and the arbitrary distance between the gap and the filler in the string generated is only due to any adjoining in this tree. The theory underlying TAG's states that the verb and all its complements are a part of the the same elementary tree, and the dependent items are either in the same elementary tree or are found in a tree and the auxiliary tree which gets adjoined in it[3].

We state equations over the feature structures of nodes of an elementary tree. For example, the identity of the agreement features of the subject and the main verb is stated in the elementary tree. This identity can thus be checked at lexical insertion and need not be done by feature passing as is the case in PATR-II, GPSG, etc. As in the case of PATR-II, where the equations associated with a rewrite rule result in a feature structure associated with the rule, equations over an entire elementary tree results in an elementary feature structure associated with this elementary tree. An example of an elementary tree with its associated feature structure (given by its graphical representation) is given below.

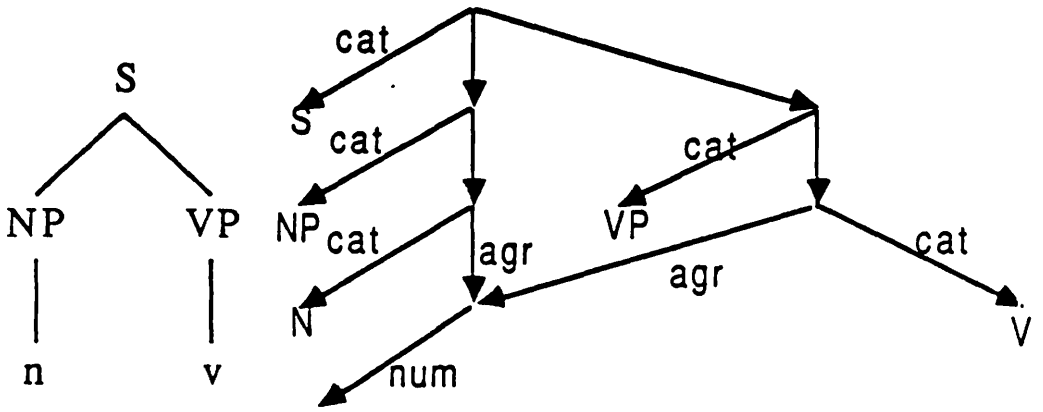

Figure 5.1: An elementary tree and associated feature structures

This example shows that inequality of agreement features of n and v nodes is known at lexical insertion and before its use in a derivation. In the derivation of the sentence

The man who eats sleep

[3]For example, this happens when there is an obligatory adjunction constraint in a tree.

we do not have to attempt the adjunction of the relative clause auxiliary tree. However, in a CFG based unificational systems only after the complete derivation (since the top S node is the first common ancestor for the two nodes) is this fact known.

5.2.2 General Schema

Any node in an elementary tree is related to the other nodes in the elementary tree in two ways which will influence the features of the node. Attribute-value or feature-value statements (F-V statements in short) about a node can be made on the basis of:

1. the relation to its siblings and the ancestor, i.e., the view of the node from the top. Let us call this F-V statement as t.

2. the relation to its descendants, i.e., the view from below. This F-V statement is written as b.

Note that both the t and b F-V statements hold of the node. Ordinarily we would expect to state one F-V statement with a node, and since both the statements, t and b, together hold for the node, we would expect to merge them (i.e., unify them). However, on adjunction, there is no longer a single node; since t is a view from the top, it must now be a statement that holds for the root of the auxiliary tree. Similarly b is now a statement about the foot node of the auxiliary tree that gets adjoined there. We believe that this approach of associating two statements with a node in the auxiliary tree is in the spirit of TAG's because of the OA constraints in TAG's. A node with OA constraints *cannot* be viewed as a single node and must be considered as something that has to be replaced by an auxiliary tree. t and b are restrictions about the auxiliary tree that must be adjoined at this node. Note that if the node does not have OA constraint then we should expect t and b to be compatible since in the final sentential tree, this node remains as a single node.

Thus, in general, with every node labeled by a nonterminal, η, (i.e.,where adjunction could take place), we associate two F-V statements, t_η and b_η. With the terminal nodes, we would associate only one F-V statement, which is unified with the F-V statement of the lexical item on lexical insertion.

Let us now consider the case when adjoining takes place as shown in the figure 5.2. The notation we use is to write alongside each node, the t and b statements. Firstly, to allow for adjunction, the root and foot nodes of the auxilary trees too will have the t and b statements associated with them. Let us say that t_{root},b_{root} and t_{foot},b_{foot} are the t and b statements of the root and foot nodes of the auxiliary tree used for adjunction at the node η. Based on what t and b stand for, it is obvious that on adjunction the statements t_η and t_{root} hold of the node corresponding to the root of the auxiliary tree (see figure 5.2). Similarly, the statements b_η and b_{foot} hold of the node corresponding to the foot of the auxiliary tree. Thus, on adjunction, we unify t_η with t_{root}, and b_η with b_{foot}. Thus, this adjunction is permissible only if t_{root} and t_η are compatible as are b_{foot} and b_η. On the other hand, if we do not adjoin at the node, η, then we unify t_η with b_η.

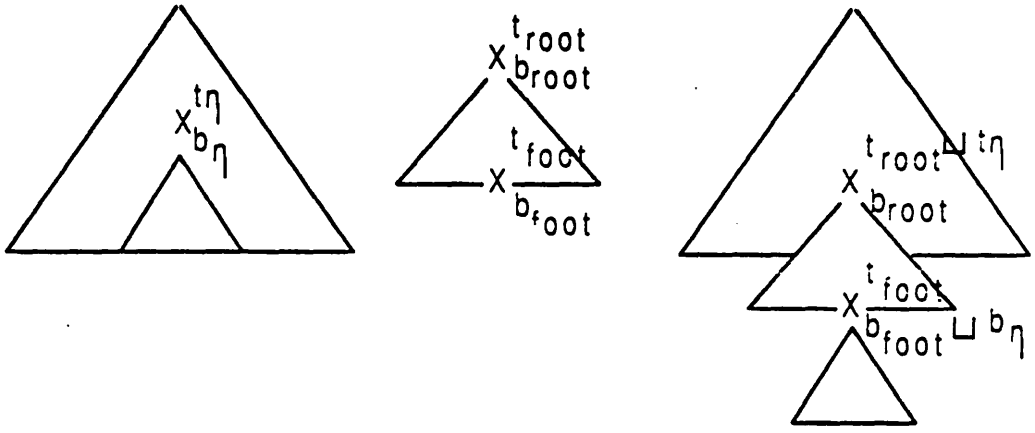

Figure 5.2: Feature structures and adjunction

We now give an example of an initial tree and an auxiliary tree and follow it up (in Section 5.2.4) with some examples of FTAG can be used.

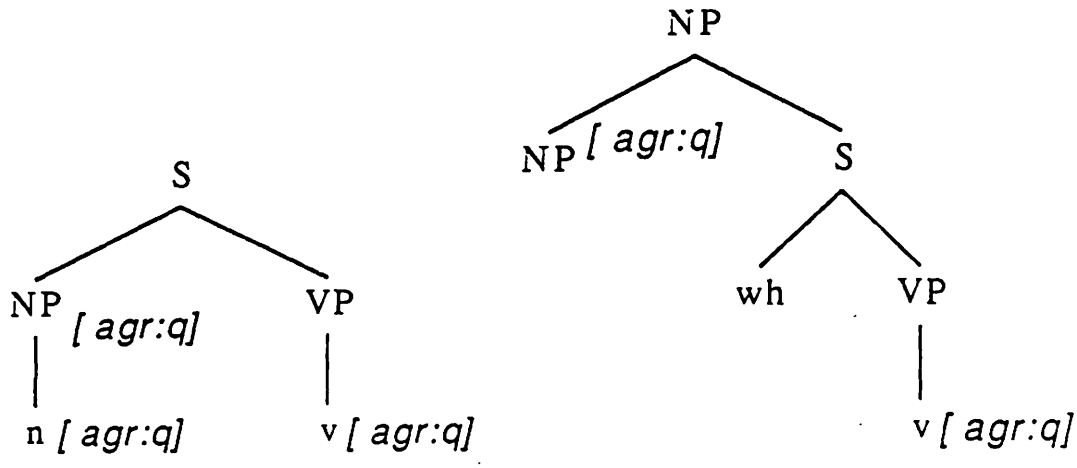

Figure 5.3: Example of feature structures associated with elementary trees

5.2.3 Adjoining and Function Application

The elementary feature structure associated with each elementary tree encodes certain relationships between the nodes. Among these relationships the sibling and ancestor/descendent relationship are included; in short, the actual structure of the tree. Thus, associated with each node is a feature structure which encodes the subtree below it. We use the attributes $i \in \mathcal{N}$ to denote the i^{th} child of a node.

Consider the adjunction sequence shown in the figure 5.2. The feature structure associated with the node where adjunction takes place should reflect the feature structure after adjunction and as well as without adjunction (if the constraint is not obligatory). Further, the feaure structure (corresponding to the tree structure below it) associated with the foot node is not known but gets specified upon adjunction. Thus, we assume that the bottom feature structure associated with the foot node is b_{foot} conjoined with a feature structure for the tree that will finally appear below this node. Since this is not known at this moment, we will treat it as a variable that gets instantiated on adjunction (taking the value of the feature structure of the subtree below the node where adjunction takes place). Thus, we can think of the auxiliary tree as corresponding to functions over feature

structures (by λ-abstracting the variable corresponding to the feaure structure for the tree that will appear below the foot node). Adjunction correponds to applying this function to the feature structure corresponding to the subtree below the node where takes place. We shall discuss this aspect later in Section 5.2.5, when we give a logical formulation of FTAG's.

5.2.4 Unification and Constraints

In earlier versions of TAG's [Joshi 75,Joshi 85], any auxiliary tree could be adjoined at a node as long as the labels of the node of adjunction and those of the root and foot nodes of the auxiliary tree used for adjunction matched. Local constraints were added to TAG's since the category symbol information was linguistically not sufficient to restrict the choice of auxiliary trees that can be adjoined at a node. For example, the subcategorization information is sometimes needed to decide the set of auxiliary trees that can be adjoined at a node. Since there are linguistic reasons determining why some auxiliary tree can be adjoined at a tree and why some cannot, or why some nodes have OA constraint, we would like to express these constraints in the F-V statements, so that local constraints (as they are expressed now) need not be stated at all.

We will now discuss how local constraints are expressed as F-V statements in FTAG. Notice, from figure 5.2, t_η and t_{root}, and b_η and b_{foot} must be compatible for adjunction to occur. We hope to specify some feature-values in these t, b statements to specify the local constraints so that

1. if some auxiliary tree is not adjoinable at a node (because of its SA constraint) then some unification involved (t_η with t_{root}, or b_{foot} with b_η) in our attempt to adjoin this auxiliary tree will fail, and

2. if node has OA constraint, we should ensure that an appropriate auxiliary tree

does get adjoined at that node.

Obligatory Adjoining Constraints

$O.A$ constraints can be easily stated in FTAG. There may be several reasons why nodes have $O.A$ constraint, which we will have to express via the F-V statements individually. However, we give a general methodology to express the constraint that adjunction is mandatory at a node.

As we have stated before, if there is no adjoining at a node, the t and b statements associated with the node have to be compatible. The reason is that this node remains as a single node in the final tree derived, and hence, the view from the top and the view from below have to be compatible. Hence, we will attempt to unify the two F-V statements. Thus, if a node has $O.A$ constraint, the only way to ensure adjunction is to have the t and b statements incompatible. Therefore, both t and b must have at least one attribute in common such that the values are not unifiable.

We now give an example of $O.A$ constraint and show how they can implemented in FTAG. This example is an oversimplification of the actual linguistic phenomenon, but will serve its purpose as an illustrative example. The node marked with an asterisk has an $O.A$ constraint because we expect adjunction with an inverted question. Thus, in the view from below, the sentential subtree (corresponding to *John saw*) is not inverted. This can be implemented naively in the following way using an attribute $inverted$ which takes values from $\{+,-\}$. The b statement for this node asserts that $inverted : -$. In the view from the top, we expect an inverted sentential component. Thus, the t assertion is $inverted : +$. If we now try to unify t and b, it will fail because the values for $inverted$ feature are not the same and hence we know that adjunction is mandatory at this node. In the auxiliary tree we know that t_{root} has to be $inverted : +$ and b_{foot} has to be $inverted : -$. Without considering further adjunctions at the root and foot nodes

and merging their t and b assertions, we see adjunction is allowed as desired.

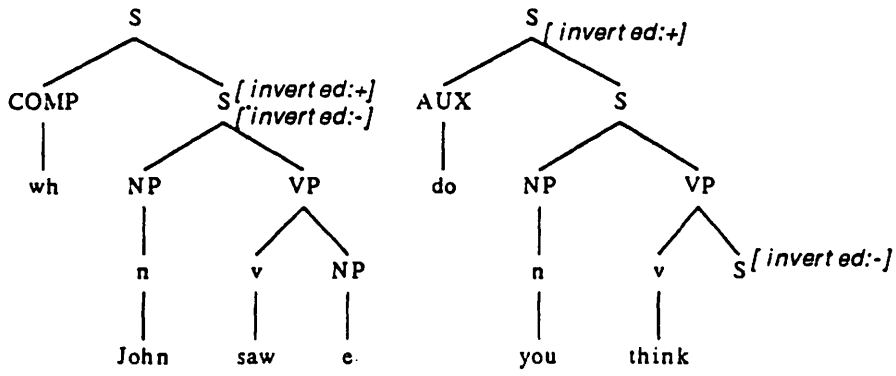

Figure 5.4: Illustration of the implementation of OA constraints

Selective Adjoining Constraints

To attempt adjunction at a node η is to try to unify t_η with t_{root} and b_η with b_{foot}. Thus, these F-V statements constrain adjoining. SA constraints are implemented by suitably specifying these statements so that an auxiliary tree not licensed by the SA constraint cannot be adjoined. We now give an example to illustrate how SA constraints can be implemented.

This example illustrates the implementation of both the OA and SA constraint. The view of the root node from below suggests that b statement for this node makes the assertion that the value of the *tense* attribute is $-$ (or untensed). However, the t statement should assert $tense : +$ (since every complete sentence must be tensed)[4]. Thus, an auxiliary tree whose root node will correspond to a tensed sentence and whose foot

[4] t statement is more complicated than just "view from the top". t statement is a statement about the node while viewing the node from the top, and hence is a statement concerning the entire subtree below this node (i.e., including the part due to an auxiliary tree adjoined at the node), and how it constrains the derivation of the nodes which are its siblings and ancestors. b remains the same as before, and is the statement about this node and the subtree below it, without considering the adjunction at this node.

node will dominate an untensed sentence can be adjoined at this node. Therefore, only those auxiliary trees whose main verb subcategorizes for an untensed sentence (or an infinitival clause) can be adjoined at the root node of this initial tree. This shows why only auxiliary tree such as β whereas an auxiliary tree corresponding to *John thinks S* can not be adjoined since the verb *thinks* subcategories for a tensed sentence.

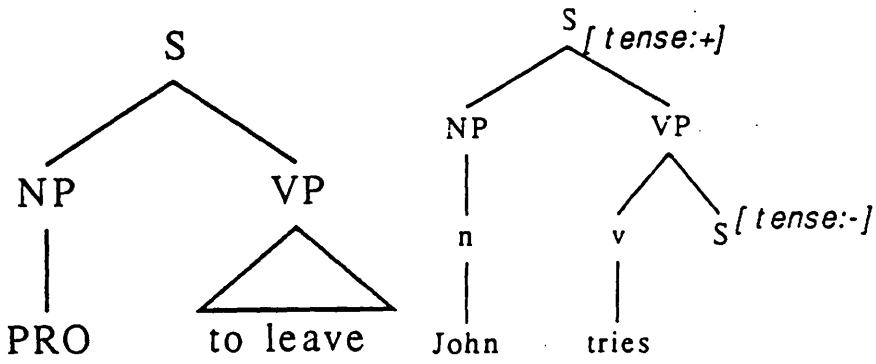

Figure 5.5: Illustration of implementation of SA constraints

Null Adjoining Constraints

NA constraints are just a special case of SA constraints and hence can, in general, be treated as SA constraints. However, some comments are in order. In section 5.3.1, we discuss a linguistic stipulation made in TAG grammars that only the foot nodes of auxiliary trees have NA constraints, and further, that all foot nodes have NA constraints. One possible way to implement this lingistic stipulation is to incorporate it in the definition of FTAG, and not state the constraints (via the features) individually with all the foot nodes. Thus, we associate only one F-V statement with the foot node of every auxiliary tree. Let us call this statement t_{foot}. We have to alter the definition of the system suitably in order to capture this stipulation. On adjunction at a node, η, we unify t_η with t_{root} as before, but unify b_η with t_{foot}. This solution is consistent then with the spirit in which TAG's are used, since the formalism itself embodies the linguistic stipulation and we

do not ever have to ever state the NA constraints. Details of the justification for this linguistic stipulation and its implementation is given in section 5.3.1.

Comments on the Implementation of Constraints in FTAG

Earlier, we had presented some reasons why local contraints had to be introduced in the TAG formalisms. Since the number of elementary trees is finite, specification by enumeration is possible. However, specification by enumeration is not a linguistically attractive solution. On the other hand, in FTAG we associated with each node two feature structures which are declarations of linguistic facts. The fact that only appropriate trees get adjoined is a corollary of the fact that only trees consistent with these declarations are acceptable trees in FTAG.

We have just seen how OA and SA constraints are implemented in FTAG. However, there are some differences in the specification of constraints in the TAG's and FTAG's. In a TAG, the local constraints specify a *finite* number of trees that can be adjoined at a node. These constraints are pre-specified and not dynamically instantiated. However, the implementation of constraining adjunctions in FTAG is quite different. In a FTAG, constraints are dynamically instantiated and are not pre-specified as in a TAG. This can be advantageous and useful for economy of grammar specification. We now give two examples, giving their implementations in a TAG and an FTAG.

In the first example, we will consider the derivations of sentences like

What do you think Mary thought John saw

In the TAG formalism, we are forced to replicate some auxiliary trees. As seen in the TAG fragment below, since the intermediate phrase *what Mary thought John saw* is not a complete sentence, we will have to use OA constraints at the root of the auxiliary tree β[5]. However, there should not be OA constraints for this tree when it is used in some

[5]For convenience, we will assume that we have implemented the stipulation that the foot nodes have

other context; for example, in the case of the derivation of

Mary thought John saw Peter

Thus, we will need another auxiliary tree with exactly the same tree structure as β_1 except for the constraints at the root node. As seen in the figure below, we can make use of the fact that constraints are dynamically instantiated and give only one specification of β_1. When used in the derivation of

What do you think Mary thought John saw

t_{root} inherits the feature $inverted : +$ which it otherwise does not have, and b_{root} inherits the feature $inverted : -$. Thus, this node, by the dynamic instantiation of the feature structure, gets an $O.A$ constraint. Note that there will not be any $O.A$ constraint in nodes of the final tree corresponding to

What do you think Mary thought John saw.

Also, the root of the auxiliary tree, corresponding to *Mary thought S*, does not get $O.A$ constraint, when this tree is used in the derivation of the sentence

Mary thought John saw Peter.

The second example illustrates the case where lexical insertion instantiates the constraints. Elementary trees having the subtree correponding to a NP node domination a N node, must not allow the adjunction of relative clauses at the NP nodes if the a pronoun is lexically inserted at the N node, but must allow adjunction if a plural noun, or a proper name is used. In a TAG, in order to implement this fact, we will need two copies of the tree for the two possiblties, whereas it can be implemented in FTAG in a straightforward manner without any duplication in the specification.

$N.A$ constraints and hence we associate only one F-V statement with foot nodes.

5.2.5 Mathematical Model of FTAG

In order to define the structures manipulated by FTAG's, we will give a mathematical model of these structures and how they are composed. The model presented below is based on that presented by Rounds and Kasper [Rounds 86] and a related model by Ohori [Ohori 87].

First we present the exact syntax we use for structures used in FTAG. The feature structures themselves are represented as formulae in Rounds-Kasper logic. We show how to extend the syntax to cope with structures that are actually used in FTAG's. Let us consider the Figure 5.2, where β is adjoined at node η in some tree γ. We adopt the convention that if f is some feature structure then it's representation in R-K logic is given by f'. Thus, the feature structure t_η will be represented by the formula t'_η, for example.

The formula for γ will be of the form

$$(\ldots t'_\eta \wedge f_\eta \wedge b'_\eta \wedge \ldots)$$

where f_η is some formula used to connect the view from the top (t'_η) and the view from below (b_η). The formula for the auxiliary tree β will be of the form

$$(t'_{root} \wedge \ldots b'_{foot})$$

The tree obtained after adjunction will then be represented by the formula

$$(\ldots t'_\eta \wedge (t'_{root} \wedge \ldots b'_{foot}) \wedge b'_\eta \wedge \ldots)$$

This can be obtained by considering (as suggested in Section 5.2.3) auxiliary trees as functions over feature structures. Under this treatment, the representation of β is a function, say f_β, of the form

$$\lambda f.(t'_{root} \wedge \ldots (b'_{foot} \wedge f))$$

To allow the adjunction of β at the node η, we have to represent γ by

$$(\ldots t'_\eta \wedge f_\beta(b'_\eta) \wedge \ldots)$$

Thus, corresponding to adjunction, we have function application. In this case, we will obtain the required formula. But note that if we do not adjoin at η, we would like to obtain for γ the formula

$$(\ldots t'_\eta \wedge b'_\eta \wedge \ldots)$$

which can be obtained by representing γ by

$$(\ldots t'_\eta \wedge I(b'_\eta) \wedge \ldots)$$

where I is the identity function. Similarly, we may have to attempt adjunction at η by any auxiliary tree (SA constraints are handled by success or failure of unification). Thus, if $\beta_1 \ldots \beta_n$ form the set of auxiliary tree we have a function, F, given by

$$F = \lambda f.(f_{\beta_1}(f) \vee \ldots \vee f_{\beta_n}(f) \vee I(f)) = \lambda f.(f_{\beta_1}(f) \vee \ldots \vee f_{\beta_n}(f) \vee f)$$

and represent γ by

$$(\ldots t'_\eta \wedge F(b'_\eta) \wedge \ldots)$$

This is our representation of feature structures used in FTAG's. We have extended R-K logic by adding λ-abstraction and application. We give the precise syntax for representing a grammar in FTAG. The same syntax as in R-K logic is used for feature structures ($e_1 \wedge e_2, l : e$, etc.) except for the clause for reentrancy. In R-K logic, this is represented by a syntactic clause of the form $\{p_1, \ldots, p_n\}$ where for $1 \leq i \leq n$, $p_i \in L^*$ represent paths which lead to the same feature structure. We use the notation $rec < x_1, \ldots, x_n >< e_1, \ldots, e_n >$ to represent reentrancy, where $x_1, \ldots, x_n$ are variables, $e_1, \ldots, e_n$ are formulae which could involve these variables. The formula $rec < x_1, \ldots, x_n >< e_1, \ldots, e_n >$ represents a set of equations given by $x_i = e_i$ for $1 \leq i \leq n$. The exact syntax we use to represent feature structures is given by those used in R-K logic except that to specify reentrancy, we use

$first(rec < x_1, \ldots, x_n >< e_1, \ldots, e_n >)$, where $first$ is the projection function picking out the first element of an n-tuple. For example, the following feature structure

$$\begin{bmatrix} cat: & NP \\ agreement: & \boxed{1}\begin{bmatrix} number: & singular \\ person: & third \end{bmatrix} \\ subject: & \boxed{1} \end{bmatrix}$$

is represented as

$$first(rec \ < x_1, x_2 > \\ < cat : NP \wedge agreement : x_2 \wedge subject : x_2, \\ number : singular \wedge person : third >)$$

Note that using this notation, we can represent any directed graph, not just acyclic ones. The motivation for using cyclic graphs has been given in [Shieber 85].

5.2.6 Modeling the structures used in FTAG *

* Please ignore this section. A completely revised version of this section will be issued as a seperate paper at a later date. Pages 141-154 have been removed.

5.2.7 Undecidability of FTAG

In this section, we show that the logical calculus that we have set up is undecidable. The proof uses the undecidability of FUG's. Rounds and Manaster-Ramer [Rounds 87] give a logical formulation of FUG's and showed that the logical system developed is undecidable by reducing it to the halting problem.

The logical formulation of FUG's, as given in [Rounds 87], has in addition to the Rounds-Kasper logic, the clause of the following form.

$$\psi \text{ where } [X_1 ::= \phi_1, \ldots, X_n ::= \phi_n]$$

where $X_i's$ are called typed variables. We will not consider other clauses which can be used to represent word order as they are not crucial for our purpose. Type variables are introduced to capture recursion in grammars. An example formula using this clause is given below.

$$S \text{ where}$$
$$S ::= (1 : a \wedge 2 : S) \vee (1 : b \wedge 2 : T)$$
$$S ::= (1 : b \wedge 2 : S) \vee (1 : a \wedge 2 : T)$$

This formula captures the following productions

$$S \rightarrow aS \mid bT$$
$$T \rightarrow bS \mid aT$$

We can encode any formula in this logic in the logic we have developed. All clauses remain the same except those using type variables. Wherever a type variable X appears, we replace it by X'(nil). Thus, if there is a subformula of the form defining the type

variable X as $X ::= \phi$, then the type variable X is replaced by a function X' which is written as $X' = \lambda f(f \wedge \phi')$ wherever ϕ' is obtained by changing every type variable Y in ϕ by $Y'(nil)$. Since we have the equivalence $\psi \wedge nil \leftrightarrow \psi$, we have encoded the formula in FUG in our logic. Since, satisfiability of FUG (its logical version) is undecidable, so is the case for the logic we have developed.

5.3 Restricted use of the Feature Based System

We have seen in Section 5.2.7 that the unrestricted use of feature structures when coupled with a grammar which allows for generation of arbitrarily large structures leads to a system which is undecidable. There are several ways of restricting the power of the system. For example, LFG assumes the *offline-parsability* constraint which places restrictions on the size of the trees generated in its *c-structure*. GPSG restricts the growth of the feature structures in a derivation. In this section, we consider a restricted version of FTAG under the assumption that a TAG with its larger domain of locality does not require a feature based unificational system in its full glory. In this restricted system (henceforth called RFTAG), the t and b F-V statements are not recursive structures. This restriction is similar to that made in GPSG. Thus, we insist that no F-V statement allows paths with recursion of labels, i.e., paths of the form $\ldots l : \ldots l : \ldots$. Furthermore, the cardinality of the set, C, of atoms is finite. Thus, the $subject$ attribute cannot be used in these F-V statements since the value for the attribute is the feature structure corresponding to the entire subject NP tree and hence is not bounded in size. We had noted earlier that feature structures correspond to the equation $F = (L \rightarrow F) + C$, where C is the set of atomic values, and L is the set of labels (or attributes). But we can give a more fine-grained characterization of feature structures. For example, a feature structure $number : o$ contains the information that ϕ is a value from the set $\{singular, plural\} \subset C$. Similarly, in the feature structure of the form $agr : \phi$, ϕ can take only certain values corresponding

to the attributes *number*, *person*, ~~gen~~. Thus, a more fine-grained equation for feature structures would be

$$F = (L_R \rightarrow F) + F_n + \ldots + F_1 + F_0 \quad \text{for some n}$$
$$F_i = L_i \rightarrow F_{i-1} \qquad 1 \leq i \leq n$$
$$F_0 = C \qquad \text{atomic values}$$

Examples of L_1 are *number*, *person*, etc., an example L_2 is the attribute agr, and examples of L_R are *subject*, *subcat*, 1, 2, etc. This approach is like giving a typing for feature structures where the feature structure corresponding to $L_R \rightarrow F$ will have a recursive type, and $F_i = L_i \rightarrow F_{i-1}$ would have a nonrecursive type. In RFTAG, we allow unification of feature structures which are of nonrecursive types (or finite types). Note, unification of nonrecursive feature structures will result in another nonrecursive feature structure.

5.3.1 A Restricted System

We now describe a restricted version of FTAG's, called RFTAG's, which allows unification of the nonrecursive feature structures only. The notation used in RFTAG is as follows. As before, we write the t and b statements alongside the nodes, using the matrix notation of the PATR-II system. However, b F-V statement is not of finite type if we consider the the attributes $\{1, 2, \ldots\}$. Since the values are obvious for these attributes are obvious from the tree structure and need not be mentioned explicitly. Leaving aside such attributes (which have recursive structures for values), we write finite type feature structures l (lower) and u (upper) along with each node. On adjunction, we unify u_{root} with u_η and l_{foot} with l_η. Note that we never have to alter the feature structures which are values of the attributes 1, 2, etc. Note all the examples we have given thus far fall under this restricted case.

It is obvious that the resulting system, i.e., RFTAG, is decidable. Since C and L are finite, we can find out for every l and u feature structure all the possible structures they could become (after unifications during a derivation). By multiplying out the different possibilities for each feature structure used, we can obtain as many trees as required and assigning constraints appropriately, we obtain an equivalent TAG.

Discussion on the Restrictions made in RFTAG system

We have defined RFTAG's in such a way that they are closer to TAG's in the way they describe the syntax of natural languages. By insisting that we can unify only feature structures of finite type, we are considering only finite number of possible trees which can be adjoined at a node. RFTAG's may be then thought of as a reformulation of the local constraints in TAG's. Like TAG's, informational content in every elementary tree is bounded and finite. Thus, like the local constraints, the feature structures and unification only describe a finite amount of information about the trees which may be adjoined at a node. However, as we have seen earlier, the terminology used in RFTAG's adds to the ability to make linguistic statements in the description of a grammar; the local constraints then are a consequence of these statements, rather than notations whose use are not self-evident (as are the SA and OA constraints). Further, the feature structures used in RFTAG's can be used to describe the co-occurence of features of dependent items and the grammatical relations that are localized within the elementary trees.

Recursive feature structures have been used in the linguistic literature. For example, an analysis of coordination in Dutch in LFG [Bresnan 82], uses unification of unbounded feature structures (encoding entire derivations). Such a device is needed if arbitrary number of pairs (or n-tuples) of dependent elements lie in different branches of a tree as in the tree structures generated for this analysis in LFG.

However, there is still questions whether such analysis are needed (see, for example,

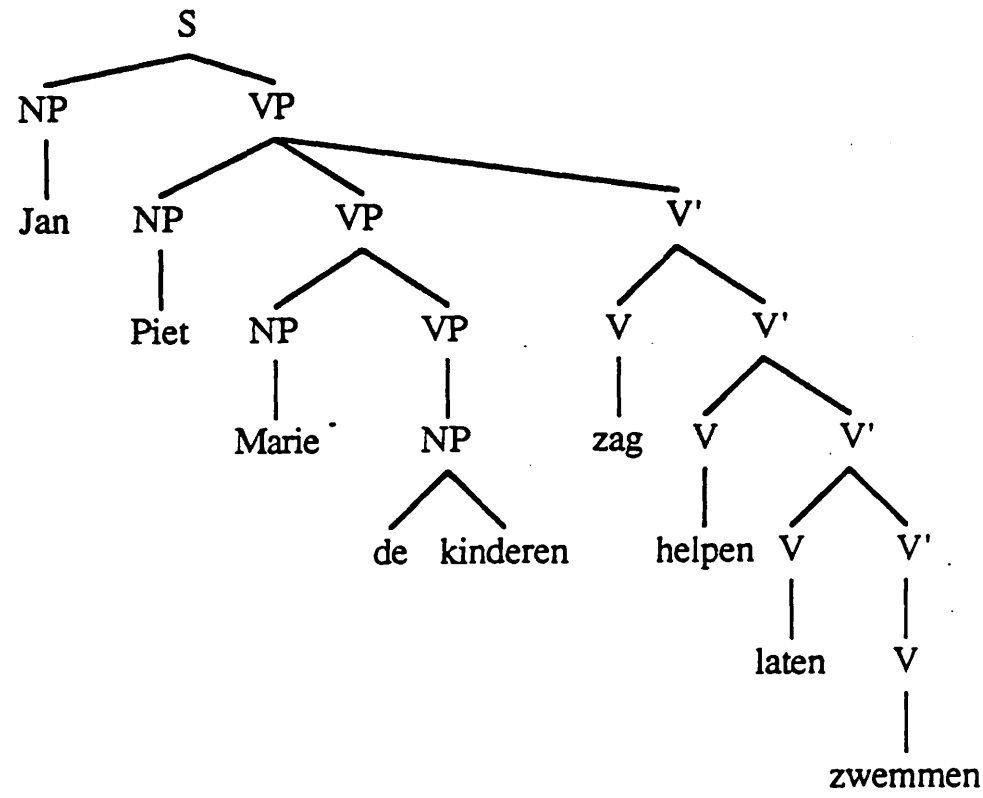

Figure 5.6: Coordination in Dutch

[Berwick 84]). Furthermore, allowing recursive types increases the power of the system drastically. Thus, it is worth questioning the use of such machinery before adopting it.

Recursive feature structures have also been used to capture the subcategorization phenomenon by having feature structures that act like stacks (and hence unbounded in size). However, in TAG's, the elementary trees give the subcategorization domain. As noted earlier, the elements subcategorized by the main verb in an elementary tree are part of the same elementary tree. Thus, with the feature structures associated with the elementary trees we can just point to the subcategorized elements and do not need devices in addition to the nonrecurive feature structures. Note, that the stacking device needed for subcategorization is given by the TAG formalism itself, in which the tree sets generated by TAG's have context free paths (unlike CFG's which have regular paths). This additional power provided by the TAG formalism has been used to an advantage in

giving an account of West Germanic verb-raising [Santorini 86].

We would like to point out a shortcoming of RFTAG's. A restriction made in RFTAG's is the cardinality of the set of atoms is finite. Under this restriction, it will not be possible to state the coindexing of NP's for example, since we will have to assume an unbounded (infinite) number of indices to account for it. This matter will be considered in future.

Some Possible Stipulations in RFTAG

In this section, we will discuss some possible stipulations in a RFTAG. However, at this stage, we do not want to consider these stipulations as a part of the formalism of RFTAG. First, some of the linguistic issues pertaining to these stipulations have not yet been settled. Secondly, our primary concern is to specify the RFTAG formalism. Further as we shall see, if the formalism has to incorporate these stipulations, it can be done so, without altering the mechanism significantly.

We said that a b feature structure is a statement made about a node when its viewed from below. In order to be consistent, we would expect that the b statement for the foot node of the auxiliary tree will be an empty statement. However, we are hesitant about making this stipulation in the definition of RFTAG until we are sure about the linguistic issues involved. The implementation or the semantics of the formalism does not change significantly even if we incorporate this stipulation into the definition.

The current linguistic theory underlying TAG's assumes that every foot node has a NA constraint. The justification of this stipulation is similar to the projection principle in Chomsky's transformation theory. It is appealing to state that the adjunction operation does not alter the grammatical relations defined by the intermediate tree structures. For example, consider the following derivation (given in figure 1.8) of the sentence

Mary thought John saw Bill hit Jill.

If the derivation results in the intermediate tree corresponding to *Mary thought Bill hit Jill*, then we would expect to obtain the relation of Mary thinking that "Bill hit Jill". This relation is altered by the adjunction at the node corresponding to the foot node of the auxiliary tree corresponding to *Mary thought S*.

If we wish to implement this stipulation, one solution is to insist that only one F-V statement is made with the foot node, i.e., the t_{foot} and b_{foot} are combined. The definition of adjunction can be suitably altered. Again, for the same reasons as above, in studying the system we do not incorporate it into to our definition.

Chapter 6

Conclusions and Future Work

In this thesis, we have studied several properties of Tree Adjoining Grammars. These results allow us to conclude that TAG s, although more powerful than CFG's, are only *mildly* so. In fact, we can realize from a number of results suggest that TAG's are a natural generalization of CFG's. Both systems are constrained in their generative capacity, with similar closure and decidability results. The class of TAL's, like the class of CFL's are both substitution closed full AFL's. Furthermore, both the formalisms have a recognition algorithm with polynomial time complexity. There are other results which point at a progression in the properties of CFG's and TAG's. We have defined an automaton, which we call embedded pushdown automaton, and show that it recognizes exactly TAL's. The epda may be thought of as a second order pda, leading to the definition of an i^{th} order pda.

The equivalence , which we establish between TAG's and Linear Indexed Grammars, is one of weak equivalence, i.e., equivalence of the string languages generated by the two formalisms. But our results indicate that it is possible to establish a stronger notion of equivalence. The tree sets of a CFG are recognized by a finite state tree automaton (defined by Thatcher [Thatcher 71]). It appears that it is possible to develop a pushdown

automaton to recognize the tree set of TAG's and LIG's. The point to note is that there is a stacking of symbols as we proceed along a path in trees generated by either system. Weir [Weir 87] has presented a hierarchy of grammatical systems which carry on this progression (of complexity of paths). It is interesting to note that this hierarchy fits in with the automaton hierarchy (as shown in [Weir 87]). There is a geometric progression in the complexity of the language hierachy, with each class in the hierarchy being a full AFL.

We have established the equivalence of TAG's with a slightly modified version of Head Grammars. The three grammatical formalisms, which we have shown to be equivalent, are apparently based on different linguistic ideas. The fact that they turn out to be equivalent suggest that these systems describe some fundamental property of natural languages. We hope that this research activity will help us in providing us fresh insight about natural languages and grammatical formalisms that describe them.

We need to study a study a stronger notion of equivalence. Since these systems have widely varying notations, it is necessary to abstract away from the details of the individual formalisms. We have made a preliminary study elsewhere [VijayShanker 87] where we characterize these systems to possess properties of context-free rewriting with linear operations. We describe a meta-level notation for such systems and conluded that any formalism that falls in this clas can be recognized in polynomial time and will generate only semilinear languages.

We have given a parsing algorithm for TAG's with polynomial time and space complexity and proved that every TAL is a semilinear language. We have also given a pumping lemma for TAL's adapting the pumping lemma for CFL's.

Finally, we have shown a method of embedding TAG's in a feature structure based framework. This system allows lingusitic statements about cooccurence of features of dependent items which are localized in elementary trees in a TAG. FTAG has several

advantages over TAG's. The specification of local constraints in a TAG is by enumeration, which is not satisfactory from the linguistic point of view. We show that in FTAG, we can avoid such specifications, instead the declarative statements made about nodes are sufficient to ensure that only the appropriate trees get adjoined at a node. Furthermore, we also illustrate how duplication of information can be avoided in FTAG's in comparison with TAG's. Finally, extensions to TAG's have been proposed that use multi-component adjoining (simultaneous adjunction of a set of trees in distinct nodes of an elementary tree) as defined in [Joshi 87b,Kroch 87]. Such extensions can be easily stated in FTAG's.

In order to make the problem of parsing this system more tractable, we propose a restriction of the use of feature structures. The resulting system, RFTAG, generates the same language as TAG's.

6.1 Future Work

There are several problems that arise out of our study of TAG's. As mentioned earlier, the equivalence of different formalisms that we have established led us to consider a stronger notion of equivalence, one in which we consider how the structures are composed in a derivation. Although the formalisms have different notations, we can verify that they share the property that rewriting is *context-free*. For example, in a TAG, the derivation trees are similar to those of a CFG. The reason for this is that there is a finite number of trees that can be adjoined at any node and this choice is pre-determined. Since the choice does not depend on the derivation sequence, we can say that the rewriting is context-free. Further the operations used in composing these structures are linear and nonerasing, i.e., no information is duplicated or lost. This led us to define a class of formalisms called linear context free rewriting systems (in [VijayShanker 86a]). We have shown, in [VijayShanker 86a], that any formalism in this class has the property that the languages generated by them are semilinear languages and can be recognized in

polynomial time . We would like to examine this class in more detail and would like to place, more precisely, the class of languages generated by them in complexity class hierarchy.

A related issue corresponds to the study of the tree sets generated by TAG's. The equivalence with Linear Indexed Grammars suggests that we could develop a pushdown based tree automaton to characterize the tree sets generated by TAG's. We conjecture that a restriction of tree pushdown automaton ([Schimpf 85,Guessarian 81]) which has linear moves would yield an automaton characterization.

We can observe that the structures composed in TAG's are done in a linear fashion. We used this property to show that every TAL is a semilinear language. It has been observed in [Joshi 85,Berwick 84] thgat this property of linear composition of bounded structures is a desirable property of formalisms used to describe natural languages. We would like to study this property and make precise this intuitive idea.

The automaton characterization of TAG's opens up a host of problems. In particular we would like to examine the deterministic epda and the parsing problem of the languages they recognize. We would also like to consider the introduction of lookaheads in the parser for TAG's.

The parsing algorithm that we have developed is an extension of the CKY algorithm ([Aho 73a]) for CFL's. We would like to examine this extension more closely and to examine whether a Valiant-style ([Valiant 75]) parser can be developed for TAG's.

We embedded TAG's in a unificational framework. We have shown that the system is too powerful, and have proposed a restriction on the use of feature structures, placing a bound on the size of feature structures used. The other restriction placed was that the set of atoms should be a finite set. We would like to relax the second restriction and allow the set to be infinite. A potential use of this relaxation is in making statements about co-indexing of nodes. We would like to examine how the power of the system is

altered by this relaxation.

The linguistic theory of feature structures is being developed. When it is developed to the point where it can be formalized, we would like to make a study of the complexity issues involved. The key problem is to determine the complexity of parsing with respect to grammar size. A formal study of the complexity issues will help us understand the extent to which the enlarged domain of locality (due to the localization of dependencies) plays a part in efficiency of parsing.

TAG's have been extended in [Joshi 87a] to allow the generation of languages with free word order. We would like to extend FTAG to allow specification of word order in a way similar to that adopted by Rounds and Manaster-Ramer [Rounds 87].

Finally, we would like to build a framework around TAG's so as to enable us to have a compositional semantics for natural languages. There have been many theories developed for expressing the semantic content of natural language constructions. The theories that we are interested in are those of the discourse representation theory of Kamp [Kamp 81] and the situation semantic theory of Barwise and Perry. Our interest in this problem is due to the question whether the localization of dependencies (for example, a predicate and all its arguments are part of the same elementary tree) and factorization of recursion will help in providing a simpler process for compositional semantics.

Bibliography

[Aho 68] Aho, A. V. Indexed Grammars — An Extension to Context Free Grammars. *J. ACM* 15:647–671, 1968.

[Aho 69] Aho, A. V. Nested Stack Automata. *J. ACM* 16:383–406, 1969.

[Aho 73a] Aho, A. V. and Ullman, J. D. *Theory of Parsing, Translation and Compiling. Vol 1: Parsing.* Prentice-Hall, Englewood Cliffs, NJ, 1973.

[Aho 73b] Aho, A. V. and Ullman, J. D. *Theory of Parsing, Translation and Compiling. Vol 2: Translation.* Prentice-Hall, Englewood Cliffs. NJ, 1973.

[AitKaci 85] Ait-Kaci, H. *A New Model of Computation Based on a Calculus of Type Subsumption.* PhD thesis, University of Pennsylvania, Philadelphia, Pa, 1985.

[Berwick 84] Berwick, R. and Weinberg, A. *The Grammatical Basis of Linguistic Performance.* MIT Press, Cambridge, MA, 1984.

[Bresnan 82] Bresnan, J. W., Kaplan, R. M., Peters, P. S., and Zaenen, A. Cross-serial Dependencies in Dutch. *Ling. Inquiry* 13:613–635, 1982.

[Gazdar 85a] Gazdar, G. *Applicability of Indexed Grammars to Natural Languages.* Technical Report CSLI-85-34, Center for Study of Lan-

guage and Information, 1985.

[Gazdar 85b] Gazdar, G., Klein, E., Pullum, G. K., and Sag, I. A. *Generalized Phrase Structure Grammars*. Blackwell Publishing, Oxford, 1985. Also published by Harvard University Press, Cambridge, MA.

[Ginsburg 66] Ginsburg, S. *The Mathematical Theory of Context Free Languages*. McGraw-Hill, New York, NY, 1966.

[Ginsburg 68] Ginsburg, S. and Spanier, E. H. Control Sets on Grammars. *Math. Syst. Theory* 2:159–177, 1968.

[Gorn 62] Gorn, S. Processors for Infinite Codes of Shannon-Fano type. In *Symp. Math. Theory of Automata*. 1962.

[Guessarian 81] Guessarian, I. On Pushdown Tree Automata. In *Proceedings of 6th CAAP*. 1981. Lecture Notes in Computer Science, Springer Verlag, Berlin.

[Joshi 75] Joshi, A. K., Levy, L. S., and Takahashi, M. Tree Adjunct Grammars. *J. Comput. Syst. Sci.* 10(1), 1975.

[Joshi 85] Joshi, A. K. How Much Context-Sensitivity is Necessary for Characterizing Structural Descriptions — Tree Adjoining Grammars. In Dowty, D., Karttunen, L., and Zwicky, A. (editors), *Natural Language Processing — Theoretical, Computational and Psychological Perspective*. Cambridge University Press, New York, NY, 1985. Originally presented in 1983.

[Joshi 86] Joshi, A. K., Vijay-Shanker, K., and Weir, D. J. *Tree Adjoining Grammars and Head Grammars*. Technical Report MS-CIS-86-1, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, 1986.

[Joshi 87a] Joshi, A. K. The Convergence of Mildly Context-Sensitive Grammatical Formalisms. 1987. Presented at the Workshop on *Processing of Linguistic Structure*, Santa Cruz.

[Joshi 87b] Joshi, A. K. An Introduction to Tree Adjoining Grammars. In Manaster-Ramer, A. (editor), *Mathematics of Language*. John Benjamins, Amsterdam, 1987.

[Kamp 81] Kamp, H. A Theory of Truth and Semantic Representation. In Groenendijk, J. A. G., Janssen, T. M. V., and Stokhof, M. B. J. (editors), *Formal Methods in Study of Languages*. Mathematical Centre Tracts, Amsterdam, 1981.

[Kaplan 83] Kaplan, R. and Bresnan, J. Lexical-functional Grammar: A Formal System for Grammatical Representation. In Bresnan, J. (editor), *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge MA, 1983.

[Karttunen 85] Karttunen, L. *D-PATR: A Development Environment for Unification-based Grammars*. Technical Report, CSLI, Stanford University, Stanford, CA, 1985.

[Kasper 86] Kasper, R. and Rounds, W. C. A Logical Semantics for Feature Structures. In 24^{th} *meeting Assoc. Comput. Ling*. 1986.

[Kay 79] Kay, M. Functional Grammar. In 5^{th} *Annual Meeting of the Berkeley Linguistics Society*. 1979.

[Kroch 85] Kroch, A. and Joshi, A. K. *Linguistic Relevance of Tree Adjoining Grammars*. Technical Report MS-CIS-85-18, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, 1985.

[Kroch 86] Kroch, A. and Joshi, A. K. Analyzing extraposition in a Tree Adjoining Grammar. In Huck, G. and Ojeda, A. (editors), *Syntax and Semantics: Discontinuous Constituents*. Academic Press, New York, NY, 1986.

[Kroch 87] Kroch, A. Subjacency in a Tree Adjoining Grammar. In Manaster-Ramer, A. (editor), *Mathematics of Language*. John Benjamins, Amsterdam, 1987.

[Ohori 87] Ohori, A. Semantics of Types for feature Structures. 1987. Manuscript.

[Palis 86] Palis, M., Shende, S., and Wei, D. S. L. An optimal linear-time parallel parser for Tree-Adjoining Languages. 1986. Manuscript.

[Parikh 66] Parikh, R. On Context Free Languages. *J. ACM* 13:570–581, 1966.

[Pereira 84] Pereira, F. C. N. and Shieber, S. The Semantics of Grammar Formalisms Seen as Computer Languages. In 10^{th} *International Conference on Computational Linguistics*. 1984.

[Pollard 84] Pollard, C. *Generalized Phrase Structure Grammars, Head Grammars and Natural Language*. PhD thesis, Stanford University, 1984.

[Roach 84] Roach, K. Formal Properties of Head Grammars. 1984. Presented at *Mathematics of Language* workshop at the University of Michigan, Ann Arbor, Oct 1984.

[Rounds 86] Rounds, W. C. and Kasper, R. .A complete Logical Calculus for Record Structures Representing Linguistic Information. In *IEEE Symposium on Logic and Computer Science*. 1986.

[Rounds 87] Rounds, W. and Manaster-Ramer, A. A Logical Version of Functional Grammar. In *Proceedings of the* 25^{th} *Annual Meeting of he*

Association of Computational Linguistics. 1987.

[Santorini 86] Santorini, B. The West Germanic Verb-Raising Construction: A Tree Adjoining Grammar Analysis. Master's thesis, University of Pennsylvania, Philadelphia, PA, 1986.

[Schimpf 85] Schimpf, K. M. and Gallier, J. H. Tree Pushdown Automata. *J. Comput. Syst. Sci.* 30:25–39, 1985.

[Schmidt 87] Schmidt, D. A. *Denotational Semantics – A Methodology for Language Development.* Allyn and Bacon. Newton, MA, 1987.

[Shieber 85] Shieber, S. M. An Introduction to Unification-Based Approaches to Grammar. Presented as a Tutorial Session 23^{rd} meeting Assoc Comput. Ling., 1985.

[Thatcher 71] Thatcher, J. W. Characterizing Derivations Trees of Context Free Grammars through a Generalization of Finite Automata Theory. *J. Comput. Syst. Sci.* 5:365–396, 1971.

[Valiant 75] Valiant, L. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.* 10:308–315, 1975.

[VijayShanker 85] Vijay-Shanker, K. and Joshi, A. K. Some Computational Properties of Tree Adjoining Grammars. In 23^{rd} *meeting Assoc. Comput. Ling.*, pages 82–93. 1985.

[VijayShanker 86a] Vijay-Shanker, K., Weir, D. J., and Joshi, A. K. On the Progression from Context-Free to Tree Adjoining Languages. In Manaster-Ramer, A. (editor), *Mathematics of Language.* John Benjamins, Amsterdam, 1986.

[VijayShanker 86b] Vijay-Shanker, K., Weir, D. J., and Joshi, A. K. Tree Adjoining and Head Wrapping. In 11^{th} *International Conference on Comput. Ling.*

1986.

[VijayShanker 87] Vijay-Shanker, K., Weir, D. J., and Joshi, A. K. Characterizing Structural Descriptions Produced by Various Grammatical Formalisms. In 25^{th} *meeting Assoc. Comput. Ling.* 1987.

[Weir 86] Weir, D. J., Vijay-Shanker, K., and Joshi, A. K. The Relationship Between Tree Adjoining Grammars and Head Grammars. In 24^{th} *meeting Assoc. Comput. Ling.* 1986.

[Weir 87] Weir, D. J. *Context-Free Grammars to Tree Adjoining Grammars and Beyond.* Technical Report, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, 1987.