A Parsing Method for Context-free

Tree Languages

Karl Max Schimpf

A DISSERTATION

in

Computer and Information Science

ented to the Graduate Faculties of the University of Penns}
artial Fulfillment of the Requirements for the Degree of Dc
osophy.

1982

rvisor of Dissertation

uate Group Chairperson

ABSTRACT

A parsing method for context-free tree languages

Karl Max Schimpf

Jean H. Gallier

Tree structures (or hierarchies) are commonly used by c
ntists. For example, data bases, theorem proving, c
riptions of abstract data types use tree structures.
ertation presents a new, more general form of tree
hing which allows one to test if a given tree fits a par
of pattern. In particular, it presents a new form
maton called a tree pushdown automaton, shows that the cl
uages recognized by tree pushdown automata is identical
s of context-free (outside-in) tree languages, and pres
er constructor for the tree pushdown automaton whic
truct a deterministic parser (called the BUTLR(0) parser)
lass of the context-free tree languages. Furthermore, the
onstructing the BUTLR(0) parser mimics LR(0) techniqu
ext-free string grammars by lifting these techniques up to
e, the BUTLR(0) parser is constructed by building a bottom-
maton, called the characteristic automaton, to recognize
characteristic trees". The characteristic automaton i
erted to a tree pushdown automaton by augmenti
acteristic automaton with internal memory in the form of
the addition of stack-like operations on these trees.

Acknowledgements

I wish to express my gratitude to Jean H. Gal
for this dissertation, for introducing me into the
Languages, for his guidance and advice throughout t
this manuscript, and for providing me with continuo
Jean has been the ideal thesis advisor. He was pat
optimistic, helpful, and full of ideas. I thank hi
advisor, but as a friend and a colleague.

I would also like to thank my committee; Pete
chairman of the committee, Aravind Joshi, and Saul
constructive criticism and suggestions. In particu
especially indebted to Saul Gorn for his careful re
manuscript and his suggestions on possible applicat

I also thank the audience of my first colloqui
strength and courage to attend (even if they felt t
their faces behind "Groucho Marx" eye glasses).

Finally, I would like to thank all my friends,
Kathy McKeown, Kathy McCoy, Sitaram Lanka, Eric May

embers of the Northwest Corridor Athletic Association, for ding unwavering interest in my work even if they did not ha test inkling of what I was doing.

Table of Contents

Chapter I

INTRODUCTION

Tree rewriting systems have been in existence fo
e some time. Among the most common and simplest
ications of tree rewriting systems is the
ax-directed translation for context-free string
uages used in compilers (see Aho and Ullman[72,79
ries[71]). However, tree rewriting systems have
used in many other types of applications. For
ance, tree rewriting systems have been used to
l formula manipulating systems such as program
mization (see LaLonde and Rivieres[81]), formula
lification (see Huet[80], Huet and Oppen[80], or
n[73]), and theorem proving (see Buchi[60], Buchi

and Elgot[58], or Elgot[61]). They are also us
abstract interpreters for recursive schemes (s*
Courcelle[76,81], Friedman[77a,77b], Gallier[8(
Nivat[75])« Yet another application of tree re
systems is to define abstract data types using
equivalence classes of trees (Guttag and
Horowitz[76,78], Wand[77], or Milner[78]).

Associated with these tree rewriting syst(
the interesting problem of recognizing if the
particular tree under consideration meets any c
"tree patterns" used by the rewrite system and,
tree matches one of these patterns, the rewrite
performs the actions associated with the "matel
pattern. Viewing this problem as a parsing prc
formal language theory, the process of testing
input tree matches any of the specified patten
viewed as performing a parse of the input tree
the topic of interest in this thesis), and the
of performing actions if a match is found can 1
as performing a transduction.

In particular, this thesis is an indepth
investigation into the development of a new foi
pattern matcher (or parser) for a common class
patterns known as context-free tree grammars.

phasis is specifically directed toward the terministic form of the parser, and a parser nerator which will construct a deterministic pars r a large subclass of the context-free tree nguages. To obtain these goals, this thesis pres new form of a tree automaton, called a tree pushd tomaton (a tree automaton augmented with internal mory consisting of a sequence of trees).

Both the tree pushdown automaton and the parse nerator for the tree pushdown automaton are inspi om LR-techniques (Knuth[68], Harrison[78], and Le d Papadimitriou[81]), and are original. The derlying concept behind the new model and the par nerator is to lift LR(0) parsing techniques for rings up to trees, in order to recognize a large bclass of the context-free tree languages using a terministic machine.

A secondary goal is to modify the tree pushdow tomaton to recognize the class of macro (or index ring languages. That is, to take the newly devel ee pushdown automaton which recognizes context-fr ee languages and apply this new parsing model to rings (using the fact that the yields of trees in ntext-free tree language corresponds to a macro language, see Fischer[68,69]). The results of t application should produce a parser whose determ form has the power to recognize a larger subclas the string languages than the class recognized b deterministic LR(k) parsers since the parsing construction method that developed the tree push automaton is a generalization of the LR(k) parsi techniques.

Before describing the type of tree pushdown automaton used in this thesis, it is important t understand what are the main issues involved. O issue is the class of context-free tree language new model will be geared to accept. Unlike context-free string languages, there are two dis classes of tree languages that can be generated context-free tree grammars (as opposed to one fo context-free string languages). The difference to the existence of two different forms of deriv (or rewrite) modes, known as inside-out and outs and each derivation mode generates a different t language (see Englefriedt and Schmidt[77,78]).

A second issue about the new model is the type
e automaton used. The two most common types of t
omata are top-down and bottom-up tree automata wh
-down automata scan the input tree from the root
frontier (leaves) while bottom-up tree automata
n the input tree from the frontier to the root (s
tcher and Wright[68], Doner[70], and Magidor and
an[69]). At first glance, this consideration may
appear to be important since it has been shown t
deterministic top-down and bottom-up tree automat
h recognize tree languages in the class of regula
e languages (Brainerd[69], Thatcher[73]). Howeve
object of this research is to develop a parser
structor to generate a deterministic parser for a
ge subclass of the context-free tree languages ar
is a well known fact that the class of regular tr
guages is identical to the class of tree language
epted by deterministic bottom-up tree automata
atcher[73]) while the class of tree languages
epted by deterministic top-down tree automata is
per subset of the class of regular tree languages
atcher[73]). Hence, in the drive for determinism
choice of tree automata may play an important ro

The third issue is what the internal memory be used for. Typically, either the memory of th parsing model is used to keep track of the porti the input already scanned or, it is used to dete what the unscanned portion of the input must loo in order for the input to be legal. To clarify difference between the two different types of us the internal memory, consider the pushdown autom which comes in LL(1) (see Lewis and Stearns[68] Knuth[71]) and LR(1) flavors (along with many ot flavors). Both the LL(1) and the LR(1) parsers pushdown automata. However, the internal memory used for very different purposes.

In a LL(1) parser, the stack is used to sim derivation in a top-down fashion. The parse sta initializing the stack to contain the start symb Then, the LL(1) parser simulates the sequence of derivation steps that generates the input string other words, at any point in time, the unscanned portion of the input string is legal if and only string the stack represents will derive (or rewr the remaining portion of the input string. Ther the LL(1) parser uses the stack to describe what unscanned portion of the input string must look order for the input string to be legal.

In a LR(1) parser, the stack is used to simulate
ation in reverse, or bottom-up fashion. At any
in time, the stack of the LR(1) parser represen
ing α such that the string α derives the porti
e input string already scanned by the LR(1) pars
he string α is legal if and only if the start
l derives the string α. Hence, the stack is us
scribe what it has already seen.

Summarizing the above issues, there appears to b
plausible models of the tree pushdown automaton
eight models are based on three major
derations and each consideration has two natural
es. These considerations are: (1) The class of
languages accepted by the model (inside-out or
de-in); (2) The type of tree automaton used
down or bottom-up); and (3) What the internal
y is used for (to describe the scanned portion o
or, to describe the unscanned portion of the
).

The model of tree pushdown automata presented in
dissertation is based on a bottom-up tree
aton, the internal memory is used to describe th

scanned portion of the input in the same manner
LR(0) parser, and is geared to recognize the cla
outside-in tree languages.

In terms of current research, only the surf
been scratched when it comes to solving the prob
parsing context-free tree languages. Most resea
focused its attention to (very) small subsets of
context-free tree languages such as regular tree
languages (see Buchi and Wright[60], Doner[70],
Eilenberg and Wright[67], Magidor and Moran[69],
Thatcher[73], and Thatcher and Wright[68]) and t
adjunct grammars (see Joshi, Levy and Takahashi[
Some research has used forms of tree grammars cl
related to context-free tree grammars such as re
schemes (see Courcelle[76,81], Friedman[77a,77b]
Gallier[80,81], and Nivat[75]). However, the em
of the research in this area has focused on the
properties of such languages as opposed to the
development of acceptors and transducers.

The only directly related research in this
has been done by Guessarian[81], which has taken
straightforward approach to the problem. Guessa
has developed a version of the tree pushdown aut
different than the one presented in this thesis,

a top-down tree automaton, which uses internal
ory to state what should appear on the remaining
canned portion of the input tree, and also
ognizes the class of outside-in context-free tree
guages. This model of tree pushdown automata
responds very closely to that of LL(0) parsers fo
text-free string languages.

The major drawback of Guessarian's model is tha
class of tree languages recognized by the
erministic version of the tree pushdown automaton
ears to be rather small. The reasons for this cl
twofold: First, it uses a top-down tree automat
its finite-state control which, as mentioned
lier, is already known to be less powerful than t
erministic bottom-up tree automaton. Secondly, t
el appears to be a generalization of the LL(0)
ser for strings (in terms of its memory usage).
well known that the class of string languages
ognized by LR(k) parsers is a superclass of the
ing languages recognized by LL(k) parsers. Hence
may assume that a generalization of the LL(0)
ser lifted to trees is not as powerful as a
eralization of the LR(0) parser lifted to trees.

Chapter 2 of this thesis provides the termi used throughout the remainder of the thesis. Th includes the definitions of sets, relations, fun strings, ranked alphabets, terms, and trees.

Chapter 3 reviews LR(0) parsing techniques. purpose of this chapter is to provide insight in use of the LR(0) parser and the construction of LR(0) parser generator. This chapters also sets background on the ideas and notions which will b in the construction of the tree pushdown automat the tree pushdown parser generator.

Chapter 4 introduces context-free tree gram and context-free tree languages. It presents th definition of a context-free tree grammar, and s how a context-free tree language is generated fr given context-free tree grammar via a series of derivation (or rewrite) steps. This includes presenting the two types of derivation modes kno inside-out and outside-in, which will generate inside-out and outside-in context-free tree lang The chapter also studies several properties of context-free tree grammars as well as transforma on these grammars to modify a context-free tree such that certain undesired properties are remov articular, one of the goals of these transformati s to introduce a standard form of context-free tr rammars called "weak Chomsky normal form".

Chapter 5 presents the main object of study, ew model of the tree pushdown automaton. Besides resenting the definition of the model, this chapt hows that for the nondeterministic version of the ushdown automaton, the class of tree languages ecognized by the model is identical to the class utside-in context-free tree languages.

Chapter 6 takes the model of the tree pushdov utomaton introduced in chapter 5, and presents a arser generator (called the BUTLR(O) parser genet hich will automatically generate a tree pushdown utomaton from a context-free tree grammar. The p onstructor is based on the notions of LR(0) parsi ifted up to trees, and produces a deterministic t ushdown automaton for a subclass of the outside-i ontext-free tree languages.

Chapter 7 takes the BUTLR(O) parser generatoi ttempts to apply this new type of parser generate he class of macro string languages. The chapter egins by showing how a context-free string gramme e lifted to a tree grammar such that the generate

BUTLR(0) parser will simulate an LR(0) parser (t showing that indeed the BUTLR(0) parser is a generalization of the LR(0) parser). It also introduces the definition of macro string gramma macro string languages. The chapter concludes b showing a possible method of using the BUTLR(0) generator to construct a new parsing model such will simulate a LR(0) parser whenever the gramma is a context-free string grammar, but more gener the sense that it is also able to also parse any language. Furthermore, it is conjectured by the that the deterministic version of this new model recognize a superclass of the string languages recognized by deterministic LR(0) parsers.

Chapter 8, the conclusion, provides a brief summary of the results of this thesis, open ques and provides a brief summary of the direction th author sees future research heading.

Chapter II

PRELIMINARY NOTATION

This chapter presents the notation and terminolog
in the remainder of this dissertation. The
pts of sets, relations, functions, and strings
nted below can be found in most elementary
matical textbooks (for further details on these
s, see Arbib, Kfoury, and Moll[81] or
er[76]).

2.1 Sets

A *set* is a collection of objects. (In this thesis, the type of sets that will frequently be consideration are *alphabets* and *languages*. A se enumerated by either listing all its members enc by braces ({a,b,c} for example) or, more general denoted $\{x \mid P(x)\}$ where $P(x)$ is a proposition describing the elements in the set. The set tha contains no elements is called the *empty set* and denoted $\emptyset$. The *cardinality* of a set A, denoted the number of elements in A. A set A is *finite* is finite, otherwise A is *infinite*. Furthermore is used to denote that "a" is a member of the se while $a \notin A$ is used to denote that "a" is not a me the set A.

Two sets A and B are *identical* (denoted A=B and only if both A and B have the same elements. A is a *subset* of the set B (denoted $A \subseteq B$) if an if every element in A is also in B. Furthermore set A is a *proper subset* (denoted $A \subset B$) if $A \subseteq$ $A \neq B$.

e union of two sets A and B (denoted $A \vee B$) is consisting of all elements that are members of set. The intersection of two sets A and B d $A \wedge B$) is the set consisting of all members of and B. The difference of two sets A and B d A-B) is the set consisting of all members of A re not members of B. The powerset of a set A d 2^A) is the set $2^A = \{B \mid B \subseteq A\}$. Furthermore, a language and B is any subset of 2^A, then B is tly called a class of languages.

e product of two sets A and B (denoted A x B) is of all ordered pairs (a,b) such that $a \in A$ and wo ordered pairs (a,b) and (c,d) are regarded as f and only if a=c and b=d. Furthermore, a over a single set A (i.e. A x A) will tly be denoted as A^2.

n-tuple of objects from a set A (denoted A)) is inductively defined as follows:

$\text{tuple}_0(A) = A$

$\text{tuple}_1(A) = \{(a) \mid a \in A\}$

iii) $tuple_{i+1}(A) = \{(a_1,\ldots,a_{i+1}) \mid (a_1,\ldots,a_i) \in tuple_i(A), a_{i+1} \in A\}$

for all $i \geq 1$

ne should note that condition (i) of the definit onstandard. Typically, $tuple_0(A)=\emptyset$. However, t implify notation later on in this thesis, the onstandard representation will be used. Let tup enote the infinite union of the sets $tuple_i(A)$ f $i \geq 0$.

.2 Relations

A binary relation (or simply a relation) is riple (A,R,B) where A and B are sets and R is a f the product A x B. Given any $a \in A$ and any $b \in B$, elated to b (denoted a R b) if and only if (a,b) imilarly, a is not related to b (denoted a $\not R$ b) nly if $(a,b) \notin R$. The domain of the relation (A,R enoted dom(R), is the set $\{a \mid a \in A, \exists b \in B$, and (a nd the range of (A,R,B), denoted range(R), is th $\{b \mid b \in B, \exists a \in A$, and $(a,b) \in R\}$. The relation (A,R, otal if A=dom(R). Also, given any relation (A,R et $R \subseteq A \times B$ denote the relation (A,R,B).

Let R $\subseteq$ A x A be a relation. R is _reflexive_ if
x for all x$\in$A, R is _transitive_ if for all x,y,z$\in$
y and y R z implies x R z, and R is _antisymmetri_
for all x,y$\in$A, x R y and y R x implies that x=y.

The _transitive closure_ of a relation R $\subseteq$ A x A
noted R^+) is the set of ordered pairs such that

i) if (a,b)$\in$R, (a,b)$\in R^+$

ii) if (a,b)$\in R^+$ and (b,c)$\in$R, then (a,c)$\in R^+$

iii) nothing else

The _transitive reflexive closure_ of a relation
A x A (denoted R^*) is the set of ordered pairs
= $R^+ \cup \{(a,a) \mid a \in A\}$.

A _partial ordering_ is any relation $\leq$ $\subseteq$ A x A su
t $\leq$ is reflexive, antisymmetric, and transitive.
relation $\leq$ is considered _total_ if and only if fo
x,y$\in$A, either x R y or y R x. Furthermore, give
partial ordering $\leq$ $\subseteq$ A x A, the _strict ordering_
ation < $\subseteq$ A x A is defined by the set of ordered
rs $\{(a,b) \mid a \leq b$ and $a \neq b\}$.

Functions

A relation $F \subseteq A \times B$ is a *partial* *function* if a
y if for all $a \in A$ and all $c,d \in B$, if a F c and a F
n $c=d$. A function is *total* if in addition dom(F)
thermore, a partial function is said to have *fini*
ain if and only if |dom(F)| is finite. For
ational convenience, a function $F \subseteq A \times B$ will be
oted as $F : A \to B$. Furthermore, if $a \in \mathrm{dom}(F)$ and
b, then b will be denoted as F(a).

Let $F : A \to B$ be any function. F is *injective*
only if for all $x,y \in A$ such that $x \neq y$, $f(x) \neq f(y)$.
surjective if and only if for all $y \in B$, there exis
$x \in A$ such that $y=F(x)$. Furthermore, F is *bijectiv*
and only if F is both injective and surjective.

Strings

Let Σ be any alphabet. The set of *strings*
noted Σ^*) is the free monoid $(\Sigma^*, \cdot, \varepsilon)$ generated b
alphabet Σ where "$\cdot$" is *concatenation* (or
taposition) and ε is the identity denoting the *em*
ing (see Lentin and Schutzenberger[67]).
thermore, let Σ^+ denote the set $\Sigma^* - \{\varepsilon\}$.

The <u>length</u> of a string $\alpha\in\Sigma^*$ (denoted length(α
string's length as a sequence of alphabet symbol
thermore, for any string $\alpha\in\Sigma^*$, α^n denotes the s
sisting of the concatenation of n sequences of th
ing α.

Let Σ be any alphabet and $\alpha,\beta,\theta\in\Sigma^*$. Then,

i) if $\alpha\cdot\theta=\beta$, then α is a <u>prefix</u> of β

ii) if $\alpha\cdot\theta=\beta$ and $\theta\neq\epsilon$, then α is a <u>proper prefix</u> of β

iii) if $\alpha=\theta\cdot\beta$, then β is a <u>suffix</u> of α

iv) if $\alpha=\theta\cdot\beta$ and $\theta\neq\epsilon$, then β is a <u>proper suffix</u> of α

e: In the literature, a string prefix (or suffix)
etimes called the head (or tail) of the string.

Natural Numbers

The set of natural numbers {0,1,2,...} is denot
N and the set of positive integers is denoted as
re $N_+=N-\{0\}$.

The function max : 2^N -> N takes a finite set
itural numbers and returns the maximal natural nui
i the set.

The function sum : 2^N ->N takes a finite set
itural numbers and returns the sura of the element
ie set•

.6 **Ranked Alphabets**

A ranked alphabet (sometimes called a stratif
r graded alphabet) is a set I> together with a rani
inction r : I. -> N. Every symbol f in I has arit;
lere n=r(f). Symbols in I. are called function syi
iere the arity denotes the number of parameters (
rguments) the function has. Symbols with arity z
re also called constants«

.7 **Terms**

A term is the structure that a macro grammar
len generating strings in the corresponding macro
inguage (see Fischer[68] and Fischer[69]). Let 2
i alphabet, | be a ranked alphabet, and X_n={x_1,..
itiote a set of n variables. The set of terms def
7 J anc* I> denoted **term(** J5,I.) , is a set of strings
lat term(l,I) C SVlV{"(",lf,!l,")fl} VX_A where

ax{r(F) | F$\in\bar{\Phi}$} and **term**($\bar{\Phi}$,$\bar{\Sigma}$) is inductively defi
follows:

i) $\varepsilon\in$**term**($\bar{\Phi}$,$\bar{\Sigma}$)

ii) if $x\in \mathbf{X}_A$, then $x\in$**term**($\bar{\Phi}$,$\bar{\Sigma}$)

iii) if $a\in\bar{\Sigma}$, then $a\in$**term**($\bar{\Phi}$,$\bar{\Sigma}$)

iv) if F$\in\bar{\Phi}$ where r(F)=0, then F$\in$**term**($\bar{\Phi}$,$\bar{\Sigma}$)

v) if F$\in\bar{\Phi}$ where r(F)=m>0 and $\alpha_1,\ldots,\alpha_m\in$**term**
then $F(\alpha_1,\ldots,\alpha_m)\in$**term**($\bar{\Phi}$,$\bar{\Sigma}$)

vi) if $\alpha,\beta\in$**term**($\bar{\Phi}$,$\bar{\Sigma}$), then $\alpha\cdot\beta\in$**term**($\bar{\Phi}$,$\bar{\Sigma}$)

vii) nothing else

rthermore, each string $\alpha\in$**term**($\bar{\Phi}$,$\bar{\Sigma}$) is called a t

Let $\bar{\Sigma}$ be an alphabet, $\bar{\Phi}$ a ranked alphabet, and
rm($\bar{\Phi}$,$\bar{\Sigma}$) be the set of terms defined by $\bar{\Phi}$ and $\bar{\Sigma}$.
ven any m-tuple of terms $(\alpha_1,\ldots,\alpha_m)$ in
ple$_m$(**term**($\bar{\Phi}$,$\bar{\Sigma}$)), and a term β in **term**($\bar{\Phi}$,$\bar{\Sigma}$), the
ring substitution of $(\alpha_1,\ldots,\alpha_m)$ into the strin
noted $\beta[\alpha_1,\ldots,\alpha_m]$, is the string subst($\beta$) wher
nction subst : **term**($\bar{\Phi}$,$\bar{\Sigma}$) -> **term**($\bar{\Phi}$,$\bar{\Sigma}$) is recursiv
fined as follows:

i) $subst(\varepsilon)=\varepsilon$

ii) $subst(x_i \cdot \theta) = \alpha_i \cdot subst(\theta)$ where $x_i \in X$

iii) $subst(x_i \cdot \theta) = x_i \cdot subst(\theta)$ where $i>m$

iv) $subst(a \cdot \theta) = a \cdot subst(\theta)$ where $a \in \overline{\Sigma}$

v) $subst(F \cdot \theta) = F \cdot subst(\theta)$ where $F \in \overline{\Phi}$ and

vi) $subst(F(\theta_1,...,\theta_m) \cdot \theta) =$
$F(subst(\theta_1),...,subst(\theta_m)) \cdot subst(\theta)$ w
and $r(F)=m>0$

In other words, every occurrence of the variab $1 \leq i \leq m$, occurring in the string β is simultaneo replaced by the string α_i.

Example 2.7.1: Let $\overline{\Sigma}=\{a,b,c\}$ and $\overline{\Phi}=\{F\}$ where r
Then, xaF(xa,yF(xb,ya,zc)b,zc)b[ax,by,cx] =
axaF(axa,byF(axb,bya,cxc)b,cxc)b.

Trees

Tree Domains -

A <u>tree</u> <u>domain</u> **D** (Gorn[62],Gorn[65]) is a nonempty
f strings over the set of positive integers $\mathbf{N}_+$
fying the following two conditions:

i) for every string u in **D**, every prefix v of u
is also in **D**

i) for every string v in **D** and every integer i i
$\mathbf{N}_+$, if the string v·i is in **D**, then for every
j in $\mathbf{N}_+$ such that $1 \leq j \leq i$ the string v·j is als
in **D**.

Essentially, a tree domain is used to provide an
ssing scheme which uniquely identifies each node
n a tree. This is achieved by letting the root o
ree have the tree address represented by the empt
g. The tree addresses of all other nodes in the
are propagated down from the root where for any
with a tree address u, its i^{th} immediate
ndant has the tree address u·i.

One of the properties of tree domains is several total orderings can be defined. In pa this thesis will use two of these orderings, t lexicographic ordering and the postfix lexicog ordering of tree domains. That is, given a tr **D**, the <u>prefix lexicographic ordering of the tr</u> **D** is the relation $\leq \subseteq \mathbf{D} \times \mathbf{D}$ such that for any addresses $u,v \in \mathbf{D}$, $u \leq v$ if and only if either

i) u is a prefix of v

ii) there exists a prefix w of u such tha $v=wjz$, $i,j \in \mathbf{N}_+$, and $i<j$

Similarly, given a tree domain **D**, a <u>postfix lexicographic ordering of the tree domain</u> **D** is relation $\leq \subseteq \mathbf{D} \times \mathbf{D}$ such that for any two tree $u,v \in \mathbf{D}$, $u \leq v$ if and only if either

i) u is a suffix of v

ii) there exists a prefix w of u such tha $v=wjz$, $i,j \in \mathbf{N}_+$, and $i<j$

2 Σ - Trees -

A Σ-<u>tree</u> (or tree for short) is a function
D -> Σ such that

i) **D** is a tree domain

ii) Σ is a ranked alphabet

ii) for every u in **D**, if $n=|\{i \in \mathbf{N}_+ \mid u \cdot i \in \mathbf{D}\}|$, th
n=r(t(u)) which is the arity of the symbol
labeling the node u

: is, a Σ-tree is a mathematical representation o
e where each node in the tree is labeled with a
ction symbol in Σ. The symbol labeling each node
: have an arity which agrees with the number of
ediate descendants the node has, and the immediat
cendants correspond to the parameters of the
ction symbol.

<u>ple</u> <u>2.8.1:</u> The tree

```
        a
      /   \
  b         c
          / | \
         b  b  b
```

defined by the function t : **D** -> Σ such that
a,b,c} where r(a)=2, r(b)=0, and r(c)=3, and the

tree domain **D** is the set D={ε,1,2,2·1,2·2,2·3}
represents the tree structure

```
      ε
    /   \
  1      2
       / | \
     21  22  23
```

The elements of the tree domain are calle
addresses. A *node* is defined as a pair (u,v)∈
where u is a tree address in **D** and v=t(u). A
(u,v) is a *leaf* if r(v)=0, otherwise (u,v) is
internal node. The node with the tree address
corresponding to the empty string is the *root*
tree. Furthermore, let (u,v) and (w,y) be any
nodes in the tree. Then, (u,v) is a *descendan*
(w,y) if w is a proper prefix of u, and (u,v)
ancestor of (w,y) if w is a proper suffix of u

Let t : **D** -> Σ be any tree. The tree dom
the tree will be denoted as dom(t). The set o
nodes, denoted leaf(t), will be the set
leaf(t) = {(u,v) | (u,v)∈t, r(v)=0}. The *dept*
tree t, denoted depth(t), is defined by the va
depth(t)=max{length(u) | u∈dom(t)}. Furthermo
any ranked alphabet Σ, let the set of all fini
Σ-trees be denoted as T_{Σ}.

It should be noted that the definitions of a

sing tree domains have existed for quite some tim

ate back to Gorn[62,65,67].

.8.3 **Trees With Variables** -

et $\mathbf{X}_n$ denote any set of n variables where

$_n = \{x_1,\ldots,x_n\}$. Adjoining $\mathbf{X}_n$ to the set of cons

n a ranked alphabet Σ (i.e. giving each variable

ank of zero), one obtains the set $T_{\Sigma}(\mathbf{X}_n)$ of trees

ariables in $\mathbf{X}_n$.

xample 2.8.2: Let Σ={a,b,c} where r(a)=2, r(b)=0,

(c)=3. Then the tree

```
      a
     / \
    b   c
       /|\
     x1 b x2
```

s a tree in $T_{\Sigma}(\mathbf{X}_2)$.

Let m be a constant in $\mathbf{N}$ and t be any tree in

$_{\Sigma}(\mathbf{X}_m)$. If $\Phi \subseteq \Sigma$, the set of nodes labeled using

ymbols in Φ, denoted $1_{\Phi}(t)$, is the set

$_{\Phi}(t) = \{u \mid (u,v) \in t,\ v \in \Phi\}$. The set of all nodes

abeled by variable symbols, denoted var(t), is th

ar(t) = $\{u \mid (u,v) \in t,\ v \in \mathbf{X}_m\}$. Furthermore, the se all nodes labeled by constant symbols in Σ, d

const(t), is the set

$\text{const}(t) = \{u \mid (u,v) \in t,\ v \in \Sigma,\ r(v)=0\}$. Note

$\text{leaf}(t) = \text{const}(t) \cup \text{var}(t)$.

<u>Note:</u> For convenience of notation, the variab

x_2, x_3, and x_4 will frequently be denoted as

and w respectively.

2.8.4 **Subtrees** -

Given a tree t and a tree address u in d

<u>subtree rooted at</u> u <u>in</u> t, denoted t/u, is the

defined by the function consisting of the set

ordered pairs

$$\{(v, t(u \cdot v)) \mid u \cdot v \in \text{dom}(t)\}.$$

In other words, it is the subtree of t, start

tree address u.

<u>Example 2.8.3:</u> Let t be the following tree:

```
        a
      /   \
   b        c
          / | \
        b   b   b
```

Then,

```
= a ,
 /   \
>       c
      / i \

     b   b   b
```

= t/21 - t/22 -" t/23 - b, and

```
 " c
>/ I \
  b  b
```

aa 2«8.1 Given any ranked alphabet i, any mM) , an 2 t€Ty(X$_m$), and any u$^{\#}$v€dom(t), (t/u)/v • t/u$^{\#}$v .

>ft Assume (w,f)€t/u*v. By the definition of trees, f»t(u$^{\#}$vw) and u* v w€dom(t) . Assume E)$(t/u)/v. By the definition of subtrees, eithe jdom(t/u) or (t/u)(Vw)^f. Similarly, either. •w$dom(t) or t(u*vw)^f. But this is a tradiction. Hence (t/u)/v <C t/u$^{\#}$v. On the other 1, assume (w,f)3(t/u)/v. By the definition of trees, vw€dom(t/u) and (t/u) (v w) = f . Similarly, •w€dom(t) and t(u$^{\#}$vw)*f. Assume (w,f)^t/u»v. B definition of subtrees, either u*vw#dom(t) or •vw)^f. But this impossible. Hence u)/v £ t/u»v, or t/u*v « (t/u)/v.

2.8.5 **Tree Replacement** -

Given a tree t_1, a tree address u i
a tree t_2, the <u>replacement of the tree</u> t
<u>subtree</u> t_1/u, denoted $t_1[u<-t_2]$, is the
the function consisting of the set of or

$\{(v,t_1(v)) \mid v \in dom(t_1)$, u is not a pr

$\cup\{(u \cdot v,t_2(v)) \mid v \in dom(t_2)\}$.

In other words, the tree t_1 is truncated
address u and the tree t_2 is inserted in

<u>**Example 2.8.4:**</u> Let t_1 and t_2 be defined

```
t1 =   a              t2 =   a
      / \                   / \
     b   c                 b   b
        /|\
       b b b
```

```
Then, t1[2<-t2] =  a
                  / \
                 b   a
                    / \
                   b   b
```

2.8.6 **Tree Composition** -

Let m be a constant in **N** and Σ be a
alphabet. Given any n-tuple of trees (t
$tuple_n(T_{\Sigma}(\mathbf{X}_m))$, and a tree t in $T_{\Sigma}(\mathbf{X}_n)$,
<u>composition</u> (or tree addition) of the fu

the functions t_1 through t_n is the tree defined l

function consisting of the set of ordered pairs

$\{(u,t(u)) \mid u \in l_{\Sigma}(t)\} \cup$

$\{(u,x_i) \mid (u,x_i) \in t,\ i>n\} \cup$

$\{(u \cdot v, t_i(v)) \mid u \in var(t),\ t(u)=x_i,\ 1 \leq i \leq n\}.$

In other words, all occurrences of the variable x

the tree t, are simultaneously replaced by the tr

Let the composition of t and $(t_1,\ldots,t_n)$ be denot

$t(t_1,\ldots,t_n)$.

Example 2.8.5:

```
Let t =  a  ,  t1 =  a  ,  and t2 = b.
        / \         / \
      x1   c      x1   x2
         / | \
       x2  b  x1

Then, t(t1,t2) = a
                /   \
               a     c
              / \   /|\
            x1  x2 b b  a
                       / \
                     x1   x2
```

Lemma 2.8.2: Given any ranked alphabet Σ, any $m \geq 0$

$p \geq 0$, any tree $t \in T_{\Sigma}(\mathbf{X}_m)$, and any two m-tuples of t

$(t_1,\ldots,t_m)$ in $tuple_m(T_{\Sigma}(\mathbf{X}_m))$ and $(s_1,\ldots,s_m)$ in

$tuple_m(T_{\Sigma}(\mathbf{X}_p))$, $t(t_1,\ldots,t_m)(s_1,\ldots,s_m) =$

$t(t_1(s_1,\ldots,s_m),\ldots,t_m(s_1,\ldots,s_m))$

Proof: Assume $t(t_1,\ldots,t_m)(s_1,\ldots,s_m) \neq$ $t(t_1(s_1,\ldots,s_m),\ldots,t_m(s_1,\ldots,s_m))$. By ins tree composition, there must exist a $u \in dom($ $v \in dom(t_i)$ for some i, $1 \leq i \leq m$, and $u \cdot v \in dom(t($ such that $t(u)=x_i$, $t_i(v)=x_j$, and $t(t_1(s_1,\ldots,s_m),\ldots,t_m(s_1,\ldots,s_m))/u \cdot v \neq s_j$ definition of tree composition and subtrees $t(t_1(s_1,\ldots,s_m),\ldots,t_m(s_1,\ldots,s_m))/u = t_i(s$ Similarly, $t_i(s_1,\ldots,s_m)/v = s_j$. But then, 2.8.1, $t(t_1(s_1,\ldots,s_m),\ldots,t_m(s_1,\ldots,s_m))/u$ which is a contradiction. Hence $t(t_1,\ldots,t_m)(s_1,\ldots,s_m) =$ $t(t_1(s_1,\ldots,s_m),\ldots,t_m(s_1,\ldots,s_m))$ is true.

2.8.7 The N^{th} m-way Tree Composition -

Let m,p be constants in **N** and Σ be a r alphabet. Given any two m-tuples of trees in $tuple_m(T_\Sigma(\mathbf{X}_p))$ and $(t_1,\ldots,t_m)$ in $tuple_m$ let the $\underline{n^{th}}$ m-way tree composition of $(t_1,\cdot$ $(s_1,\ldots,s_m)$, denoted $[(t_1,\ldots,t_m)(s_1,\ldots,s_m$ recursively defined as follows:

i) $[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^0 = (s_1,\ldots,s_m)$

i) for all $i\geq 0$, $[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^{i+1} =$
$(t_1[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^i, \ \ldots \ ,$
$t_m[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^i)$

le 2.8.6:

```
1 = a  , t  = f , s  = f , and s  = b.
   / \     2  |    1  |         2
  x   y       y       x

[(t ,t )(s ,s )]^0 = (f , b),
   1  2   1  2        |
                      x

[(t ,t )(s ,s )]^1 = (a , f), and
   1  2   1  2       / \  |
                    f   b b
                    |
                    x

[(t ,t )(s ,s )]^2 = (a , f).
   1  2   1  2       / \  |
                    a   f f
                   / \  | |
                  f   b b b
                  |
                  x
```

Chapter III

LR PARSERS

This chapter reviews a construction method build deterministic bottom-up parsers for a lar subclass of the context-free (string) grammars. parsers are called LR parsers because they scan input from left to right and construct a rightm derivation in reverse. Furthermore, it is a we fact that of all the deterministic string parse class of LR parsers recognize the largest class context-free languages (see Knuth[68]).

ogically, an LR parser consists of two parts, a routine and a parsing table (see figure 3.1.1). rsing tables are dependent on the given t-free grammar, and must be constructed, while iver routine is the same for the type of LR used. Furthermore, the construction method for ng the parsing tables is dependent on the type of ser one is interested in using, and there are l different types of LR parsers to chose from , LR(k), SLR(1), and LALR(1) to name a few). hat this chapter will only concentrate on the parser construction method (For further ation on the different types of LR parsers, and corresponding construction methods, see 68], DeRemmer[69][71][72], Harrison[78], 77a][77b], Anderson, Eve, and Horning[72], kis and Ripley[77], LaLonde, Lee, and g[71], Geller and Harrison[77], Schimpf[81], [79], and Harrison and Havel[73]).

he purpose of this chapter is not to provide us definitions and proofs about the LR(0) uction method. Instead, the intent is to provide t and background into the method of constructing parsers which will be lifted to tree grammars in ding chapters. Hence, only pertinent definitions

```
                      +-----------+
                      |  table    |     parsi
string grammar -->|           |-->
                      | generator |     table
                      +-----------+

        a) generating the parsing tables

     +----------------------
     | I n p u t   t a p e ...
     +----------------------
         ↑
         +-----------+     +---------+
         | Driver    |     | parsing |
         |           |---->|         |
         |   Routine |     | tables  |
         +-----------+     +---------+
         |
+-------+ |
| stack |<+
+-------+

            b) operation of LR parser
```

Figure 3.1.1: Layout of an LR parser

and theorems will be given. Furthermore, in ge proofs will not be provided unless they provide or construction methods in building LR(0) parse

However, before describing the LR(0) parse chapter begins by presenting background about context-free grammars, finite-state automata, a pushdown automata.

1 **Context-free Grammars**

A <u>context-free</u> <u>grammar</u> (or simply string gram
is a quadruple $G = (\Phi, \Sigma, P, S)$ where

Φ is a finite alphabet of <u>nonterminal</u> <u>symbols</u>

Σ is a finite alphabet of <u>terminal</u> <u>symbols</u>,

P is a finite set of pairs $(A,\beta) \in \Phi \times (\Phi \vee \Sigma)^*$
called <u>productions</u>, and

$S \in \Phi$ is a nonterminal called the <u>start</u> <u>symbol</u>.
production (A,β) will be denoted as $A \to \beta$. Also,
ll be assumed that all string grammars are <u>augme</u>
at is, there is a production of the form $S \to S' \in P$
alled the <u>start</u> <u>production</u> where $S, S' \in \Phi$ and S doe
ccur in any other production in P.

For notational convenience, upper case letter
ill be used to denote nonterminal symbols, lower
etters to denote terminal symbols, underlined upp
ase letters to denote grammar symbols (i.e. symb
n either Φ or Σ), lower case greek letters to den
trings of grammar symbols (strings in $(\Phi \vee \Sigma)^*$), a
he symbol ϵ will be reserved to denote the empty
tring.

sample 3«1.1: A string grammar which generates st
E the form $a^n b^n$ is G_1 «($,>,, P,S) where

Φ = {S,A};

Σ « {a,b}; and

P = {S->A, A->6, A->aAb}.

)te: S->A is the start production.

.1.1 **Derivations** -

Given a string grammar G=((J5,>, P,S), let the
le-step derivation (or rewrite) relation
=> C (UJVi)* x (fVl)* be defined by the set of pa
{(ocAjJ, oc6j3) | A€$; oc, j3 >QG(Q\/X)* ; and A-}θ€Σ
i other words, given any string ocA3 and the prod
->6, the nonterminal A can be replaced by the str
• Let —> and => denote the transitive and the
ransitive reflexive closures of => respectively.

From the above relation, the one-step rightmo
arivation relation can be defined which implies a
rdering on the rewrite steps. That is, the one-s
ightmost derivation relation $\underset{R}{\Longrightarrow}$ £ $(fVi)^*$ x (ΦVΣ
s defined by the set of pairs

{(ocAj3, oc8j3) | ocAj}=>ocejJ and $\beta \in \Sigma^*$}.

n other words, $\underset{R}{\Longrightarrow}$ is the one-step derivation app
o the rightmost nonterminal occurring in the stri

$\alpha A\beta$. Let $\overset{+}{\underset{R}{\Longrightarrow}}$ and $\overset{*}{\underset{R}{\Longrightarrow}}$ denote the transitive and transitive reflexive closures of $\underset{R}{\Longrightarrow}$ respectivel

Example 3.1.2: Let $G_2=(\Phi,\Sigma,P,S)$ be a string gram where

Φ = {S,A,B,C};

Σ = {a,b}; and

P = {S->C, C->ACB, C->ε, A->a, B->b}.

Then, aaCBB $\Longrightarrow$ aaBB using C->ε and aaCBB $\overset{*}{\Longrightarrow}$ aa using the productions C->ε, A->a, and B->b. Als aaCBB $\underset{R}{\Longrightarrow}$ aaCBb while aaCBB $\underset{R}{\not\Longrightarrow}$ aaBB since C is rightmost nonterminal in aaCBB. Also, like G_1 i example 3.3.1, the language generated by G_2 is s of the form $a^n b^n$.

3.1.2 Language Generated By Context-free Gramma

Given a string grammar $G=(\Phi,\Sigma,P,S)$, the str language generated by G, denoted L(G), is the se terminal strings derivable from the start symbol That is,

$$L(G) = \{\beta \mid S \overset{*}{\Longrightarrow} \beta,\ \beta \in \Sigma^*\}$$

Note: It can be shown that the order in which t derivation steps are applied (i.e. the choice o nonterminal to rewrite next) has no effect on th sulting string produced. Hence the language L(G) uld have been alternatively defined by the set

$$L(G) = \{\beta \mid S \overset{*}{\underset{R}{\Longrightarrow}} \beta,\ \beta \in \Sigma^*\}$$

thermore, any string of grammar symbols derivabl om the start symbol S, under a rightmost derivati a <u>sentential form</u>. That is, any string $\alpha \in (\Phi \vee \Sigma)$ sentential form if and only if $\alpha \in \{\beta \mid S \overset{*}{\underset{R}{\Longrightarrow}} \beta\}$.

<u>ample 3.1.3:</u> Let G_2 be the string grammar defined ample 3.1.2. The language generated by G_2 is the

$$L(G) = \{a^n b^n \mid n \geq 0\}.$$

example, a derivation which generates the strin abbb is as follows:

$$S \underset{R}{\Longrightarrow} C \underset{R}{\Longrightarrow} ACB \underset{R}{\Longrightarrow} ACb \underset{R}{\Longrightarrow}$$
$$AACBb \underset{R}{\Longrightarrow} AACbb \underset{R}{\Longrightarrow} AAACBbb \underset{R}{\Longrightarrow}$$
$$AAACbbb \underset{R}{\Longrightarrow} AAAbbb \underset{R}{\Longrightarrow} AAabbb \underset{R}{\Longrightarrow}$$
$$Aaabbb \underset{R}{\Longrightarrow} aaabbb$$

.3 Reduced String Grammars -

A string grammar $G=(\Phi,\Sigma,P,S)$ is <u>reduced</u> if and ly if for every production $A \rightarrow \alpha \in P$, there exists rivation such that $S \overset{*}{\underset{R}{\Longrightarrow}} \theta A \beta \underset{R}{\Longrightarrow} \theta \alpha \beta \overset{*}{\underset{R}{\Longrightarrow}} \delta\beta$ where $(\Phi \vee \Sigma)^*$ and $\delta, \beta \in \Sigma^*$. In other words, for every oduction p there exists a terminal string $\delta\beta$, in guage generated by G, such that the production p i... in the derivation producing the string $\delta\beta$ (i.e. unnecessary productions are removed).

rem 3.1.1: (see Bar-Hillel, Perles and Shamir[61] Harrison[78]) Given any string grammar $\Phi,\Sigma,P_1,S)$, one can construct a reduced string mar $G_2=(\Phi,\Sigma,P_2,S)$ such that $P_2 \subseteq P_1$ and $)=L(G_2)$.

ple 3.1.4: Let $G_3=(\Phi,\Sigma,P_3,S)$ be a string grammar e

Φ = {S, A, B, C, D, E};

Σ = {a,b,c}; and

```
P3 = {S -> C,
      C -> ACB,
      C -> ε,
      A -> a,
      B -> b,
      D -> aacbb,
      C -> E,
      E -> EcE}.
```

language generated by G_3 is the set

$L(G_3) = \{a^n b^n \mid n \geq 0\}$.

reduced string grammar of G_3 is the string gramma (Φ,Σ,P_4,S) where

$$
\begin{aligned}
P_4 = \{S &\rightarrow C,\\
C &\rightarrow ACB,\\
C &\rightarrow \varepsilon,\\
A &\rightarrow a,\\
B &\rightarrow b\}.
\end{aligned}
$$

Note: The production D -> aacbb has been remo
the nonterminal D is not reachable from the st
symbol and the productions C -> E and E -> EcE
been removed since the nonterminal E can not d
terminal string in $\overline{\Sigma}^*$.

3.1.4 Right-linear Grammars -

A string grammar $G=(\overline{\Phi},\overline{\Sigma},P,S)$ is right-lin
and only if every production $p \in P$ is of the for
where $A \in \overline{\Phi}$, $\alpha \in \overline{\Sigma}^*$, and $B \in \overline{\Phi} \cup \{\varepsilon\}$. That is, if a
production's right-hand side contains a nonter
the nonterminal must be the last symbol occurr
the right-hand side. A string grammar $G=(\overline{\Phi},\overline{\Sigma},$
strict right-linear if and only if every produ
is of the form $A \rightarrow aB$ where $A \in \overline{\Phi}$, $a \in \overline{\Sigma} \cup \{\varepsilon\}$, and
That is, a string grammar is strict right-line
right hand side of a production is either the
string, a single terminal symbol, a single non
symbol, or a single terminal symbol followed b
single nonterminal symbol.

xample 3.1.5: Let $G_5=(\Phi,\Sigma,P,S)$ where

$\Phi = \{S,A,B\}$;

$\Sigma = \{a,b\}$; and

$P = \{S\to A,\ A\to \varepsilon,\ A\to aB,\ B\to b,\ B\to bA\}$.

hen, G_5 is a strict right-linear string grammar a
enerates the language $L(G_5) = \{\alpha^n \mid \alpha=ab,\ n\geq 0\}$.

he following results about right-linear grammars
asily be shown.

heorem 3.1.2 Given any right-linear string gramma
here exists a strict right-linear string grammar
uch that $L(G_1)=L(G_2)$.

heorem 3.1.3: The class of right-linear string
rammars is identical to the class of regular
anguages.

roof: See Harrison[78] or Bar-Hillel and Shamir[6

.2 Finite-state Automata

This section presents a brief review of
inite-state automata (For more information on
inite-state automata see Harrison[78], Eilenberg[
abin and Scott[59], and Salomaa[69,73]). Logical
inite-state automaton (FSA for short) consists of
nput tape and finite-state control (see figure 3.

where the input tape is read from left to right, scanning the input tape just once.

```
+---------------------------
| I n p u t   t a p e ...
+---------------------------
     ^
     |
   +-------------+
   | finite      |
   |             |
   |    state    |
   |             |
   |     control |
   +-------------+
```

Figure 3.2.1: Layout of a finite state automat

More formally, a <u>finite-state</u> <u>automaton</u> is a quintuple $M=(\Sigma,K,\delta,q_0,Q)$ where

Σ is a finite alphabet of <u>input</u> <u>symbols</u>,

K is a finite set of <u>states</u>,

$\delta : K \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^K$ is a function called the <u>transition</u> <u>map</u>,

$q_0 \in K$ is the <u>start</u> <u>state</u>, and

$Q \subseteq K$ is a set of <u>final</u> <u>states</u>.

An _instantaneous description_ (ID for short) ovides a "snapshot" description of the FSA betwee ves defined by the transition map δ. That is, an stantaneous description is a pair $(q,\alpha) \in K \times \Sigma^*$ ere q is the _current state_ of the FSA, and α is ring left to scan. The initial configuration of A is the instantaneous description (q_0,α) where e input string to parse.

The _computation_ relation $\vdash \subseteq ID \times ID$ describe e manner in which the FSA operates. That is, giv A $M=(\Sigma,K,\delta,q_0,Q)$ and two instantaneous descriptio $_1$ and id_2, $id_1 \vdash id_2$ if and only if $id_1 = (q_1,a$ d $id_2 = (q_2,\alpha)$ where $a \in \Sigma \cup \{\varepsilon\}$ and $q_2 \in \delta(q_1,a)$. I her words, each move is a shift-move (or read-mov ere the read head on the input tape is advanced, ross the string "a", and the state is updated to

A finite-state automaton M accepts (or parses) ring α if there is a computation which will reac d of the input string α, and the corresponding rrent state of the computation is a final state. at is, the _language accepted by a finite-state tomaton M_, denoted L(M), is the set

$$L(M) = \{\alpha \in \Sigma^* \mid (q_0,\alpha) \vdash^* (q_F,\varepsilon),\ q_F \in Q\}$$

ere $\vdash^*$ is the transitive reflexive closure of $\vdash$

A FSA $M=(\Sigma,K,\delta,q_0,Q)$ is <u>deterministi</u>

if for each $q \in K$ and $a \in \Sigma$, either

i) $\delta(q,a)=\emptyset$ and $\delta(q,\varepsilon)=\emptyset$,

ii) $\delta(q,a)$ is a singleton set and $\delta($

iii) $\delta(q,a)=\emptyset$ and $\delta(q,\varepsilon)$ is a singlet

In other words, a FSA M is deterministic instantaneous description id_1, the automa defined such that there is at most one le instantaneous description (i.e. if $id_1 \vdash$ id_2 is unique).

<u>Example 3.2.1:</u> Let $M_1=(\Sigma,K,\delta,0,\{2\})$ where

$\Sigma = \{a,b\}$;

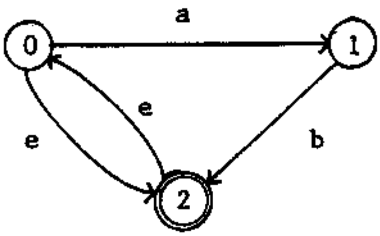
$K = \{0,1,2\}$; and

δ is defined by the following table wh input pair (c,d), the rows represent v the columns represent values of d. Fu empty table entries represent null set

	a	b	ε
0	{1}		{2}
1		{2}	
2			{0}

Note: The transition map δ can also be graphical

depicted as follows:



where the final state (i.e. state 2) is enclosed

a double circle.

The language accepted by M_1 is the set

L(M_L) - {w^{11} I w»ab, nX>}

For example, the string "ababab" is accepted as

follows:

(0,ababab) K (l,babab) h (2,abab)

f- (0,abab) I- (l,bab) I- (2,ab)

I- (0,ab) I- (l,b) t- (2,6)

which is the accepting condition. Also note that

is not deterministic since δ(0,6)=^{s}{2} and

4(0$_f$a)-{l}.

While there are many results known about

Lite-state automata, the remaining portion of thi*

ition presents only those facts which apply to LR

•sing.

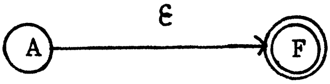
Theorem **3.2.1:** (see Harrison[78], Eilenberg[74], Salomaa[73]) The class of (string) languages acce by finite-state automata and the class of regular (string) languages are identical.

Theorem **3.2.2:** For every right-linear string gram $G=(\Phi,\Sigma,P,S)$, there exists a FSA M such that L(G)=

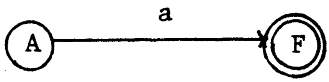
Proof: By theorem 3.1.3, the language generated b right-linear string grammar is regular. By theor 3.2.1, the class of regular languages is identica the class of languages accepted by FSA. Hence, t must exists some FSA M such that L(G)=L(M).

While the last theorem does not provide a constructive proof, one can easily build a FSA M that L(G)=L(M). That is, using theorem 3.1.3 whi states that for the right-linear string grammar $G=(\Phi,\Sigma,P,S)$ there exists a strict right-linear st grammar $G'=(\Phi,\Sigma,P',S)$, let $M=(\Sigma,\Phi\cup\{F\},\delta,S,\{F\})$ wh $F\notin\Phi$ and δ is defined such that for each productio $p\in P'$,

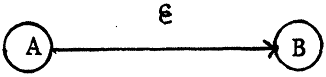
i) if $p=A\text{->}\varepsilon$, then $F\in\delta(A,\varepsilon)$



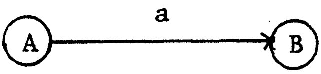
ii) if $p=A\text{->}a$ where $a\in\Sigma$, then $F\in\delta(A,a)$



iii) if $p=A\text{->}B$ where $B\in\Phi$, then $B\in\delta(A,\varepsilon)$



iv) if $p=A\text{->}aB$ where $a\in\Sigma$ and $B\in\Phi$, then $B\in\delta(A,a$



ample **3.2.2:** Let $G_1=(\Phi,\Sigma,P,S)$ where

$\Phi = \{S,A,B\}$;

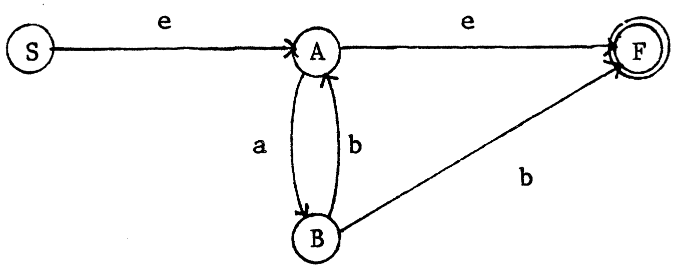
$\Sigma = \{a,b\}$; and

$P = \{S\text{->}A,\ A\text{->}\varepsilon,\ A\text{->}aB,\ B\text{->}b,\ B\text{->}bA\}$.

ıen, the corresponding FSA is $M_2=(\Sigma,\Phi\cup\{F\},\delta,S,\{F\}$

ıere the transition map δ is graphically depicted

ıllows:



Theorem **3.2.3:** (Rabin and Scott[59] and Harrison
For every FSA $M=(\Sigma,K,\delta,q_0,Q)$, one can construct
deterministic FSA $M'=(\Sigma,K',\delta',q_0',Q')$ such that
$L(M)=L(M')$ and M' does not contain any epsilon m
(i.e. for all $q \in K'$, $\delta'(q,\varepsilon)=\emptyset$).

The idea used to construct the FSA M' is to
simultaneously follow every possible computation
by having each state $q' \in K'$ be a set of states in
where q' is reachable by M' if and only if for a
$q \in q'$, q is reachable in M. The construction met
given by algorithm 3.2.1 (see below) and contain
procedures. The function "closure" takes a stat
$q' \in K'$, and returns the set of all states, reacha
from states in q', without reading any more inpu
performs epsilon closure on M). The procedure "
is the main routine. It starts by defining q_0' a
epsilon closure of the start state q_0 in M (i.e.
set of all states $q \in K$ such that $(q_0,\alpha) \vdash^* (q,\alpha$
Then, using the function "GOTO", it takes each s

K' already built, and determines the transitions
n q_1 as follows:

For each $a \in \Sigma$, if there exists $q \in q_1$ such that
$q' \in \delta(q,a)$, then there is a unique transition in M
such that $\delta'(q_1,a)=q_2$ where q_2 is the epsilon
closure of the set $\{q' \mid q' \in \delta(q,a),\ q \in q_1\}$.

e the graph defining the transition map δ' is bui
set of final states F' is defined such that for
ry state $q' \in K'$, if there exists a state $q \in q'$ such
t $q \in F$, then $q' \in F'$.

orithm 3.2.1: A method for constructing
a deterministic finite automaton.

ut: a FSA $M=(\Sigma,K,\delta,q_0,F)$ (possibly nondeterminist
put: a deterministic FSA $M'=(\Sigma,K',\delta',q_0',F')$ where
does not contain any epsilon moves.

hod: The three procedures below, initiated
by calling ITEMS(M);

cedure ITEMS(M);
begin
    for all input pairs $(a,b) \in K \times (\Sigma \cup \{\varepsilon\})$
        let $\delta'(a,b)=\emptyset$;
    q_0' := closure($\{q_0\}$);
    K' := $\{q_0'\}$;

```
      repeat
         for each set q_1∈K', and each inp
            a∈Σ such that q_2=GOTO(q_1,a) a
         do
            K' := K' ∪ {q_2};
            δ'(q_1,a) := {q_2};
         od;
      until no more sets of states can be
      F' := ∅;
      for each q'∈K' do
         if there exists a q∈q' such that
            then F' := F' ∪{q'}
         fi;
      od;
   end;
Function GOTO(q_1,a);
   begin
      q_2 := {q' | q'∈δ(q,a), q∈q_1};
      return closure (q_2);
   end;
```

```
Function closure(q);
   begin
      s := q;
      while there exists a state p∈s such that
         q'∈δ(p,ε) and q'∉s do
         s := s V {q'};
      od;
      return s;
   end;
```

Example 3.2.3: Consider the FSA M_2 created in exa 3.2.2. Using the above algorithm, the created deterministic FSA is the FSA $M_3=(\Sigma,K',\delta',q_0',Q')$ v

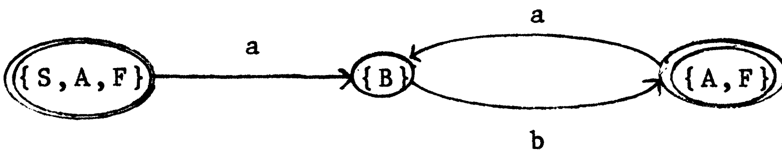
Σ = {a,b};

K = {{S,A,F},{B},{A,F}};

q_0' = {S,A,F};

Q = {{S,A,F},{A,F}}; and

δ', the transition map, is defined by the foll graph:



Theorem 3.2.4: For every right-linear stri
$G=(\Phi,\Sigma,P,S)$ there exists a FSA M such that
and M does not contain any epsilon-moves.

Proof: First, using theorem 3.2.2, one can
M′ such that L(M′)=L(G). Then, using theo
one can construct a deterministic FSA M su
L(M)=L(M′)=L(G) and M does not contain any
epsilon-moves.

Finally, one can define the relations
the sequence of states visited in a FSA, a
corresponding input string parsed. This i
the "spelling" and is defined as follows:

Definition 3.2.1: Given a FSA $M=(K,\Sigma,\delta,q_0,$
spelling : $K^* \to 2^{\Sigma^*}$ be a function recursi
such that for any string of states $s_1 s_2 \ldots$

i) spelling(ε) = $\emptyset$

ii) spelling(s_1) = $\{\varepsilon\}$

iii) spelling($s_1 \ldots s_n$) = $\emptyset$ if $n \geq 2$, and
does not exist a symbol $a \in \Sigma$ such
$s_2=\delta(s_1,a)$ or spelling$(s_2 \ldots s_n)=\emptyset$

$spelling(s_1 \ldots s_n) = \{a\beta \mid \beta \in spelling(s_2 \ldots s_n),$
$s_2 \in \delta(s_1, a)\}$ otherwise.

shdown Automata

is section presents a brief review of pushdown a (PDAs for short). One should note that the f the PDA presented in this section is not d. Instead, the model is purposely defined to resemble the type of pushdown automata used by arsers (for a more formal description of PDAs, rison[78], Oettinger[61], and Lewis and itriou[81]).

gically, a pushdown automaton consists of an ape, finite-state control, and internal memory form of a stack (see figure 3.3.1). Like state automata, the input tape is read from left t, and scanned just once. The stack is defined st-in first-out structure in which only the top can be read. Furthermore, elements can only be pushed) or deleted (popped) from the top of the and these modifications are bounded (i.e. only e sequence may be pushed or popped at a time). venience, the stack will be defined as a string ols where $\perp$ is a reserved symbol denoting the empty stack, concatenation is the operator which performs a push, and the top of t stack is assumed to be the rightmost symbol in t string.

```
                         +---------------------------
                 +-->(Input    tape...
                 I   +___________________________
              +-I------------+
              I  finite      |
              I    state     |
              j     control  j
              +--------------+
                             I
                             |
  +-----------------+        
  I Stack of symbols I<--+
  +-----------------+
```

Figure 3.3.1: Layout of a Pushdown automato

Definition 3«3.1 : A pushdown automaton (PDA) is quadruple D-(I, p, δ, J_) where

I is a finite alphabet of input symbols;

P is a finite alphabet of stack symbols;

$6 : (P \times I)\ V (P^{+} \times \{6\}) \to 2^{1}$

is a function called the transition map wt has finite domain; and

J_€p is a reserved constant denoting the empty tree stack.

The transition map δ is defined such that all initions are one of the following two forms:

<u>**ft-move:**</u>

$A \in \delta(B,a)$ where $A,B \in \Gamma$ and $a \in \overline{\Sigma}$

<u>**uce-move:**</u>

$BA \in \delta(B\alpha,\varepsilon)$ where $B \in \Gamma$, $A \in \Gamma \vee \{\varepsilon\}$, and $\alpha \in \Gamma^*$

An <u>instantaneous</u> <u>description</u> of a pushdown omaton (ID for short), provides a "snapshot" cription of the pushdown automaton between moves. t is, an instantaneous description is a pair $\alpha) \in \Gamma^* \times \overline{\Sigma}^*$ where β is the current stack and o string left to scan on the input tape. The init figuration of a PDA is the instantaneous descript $\alpha)$ where α is the input string to parse.

The <u>computation</u> relation $\vdash \underline{C}$ ID x ID describes manner in which a pushdown automaton functions. t is, given a PDA $D=(\Sigma,\Gamma,\delta,\underline{|})$ and two instantane criptions id_1 and id_2, $id_1 \vdash id_2$ if and only if the two following conditions hold:

i) $id_1 = (\beta B, a\alpha) \vdash (\beta BA, \alpha) = id_2$ where $A \in \delta($ $\beta \in \Gamma^*$, and $\alpha \in \overline{\Sigma}^*$

ii) $id_1 = (\beta B\theta, \alpha) \vdash (\beta BA, \alpha) = id_2$ wher
$BA \in \delta(B\theta, \varepsilon)$, $\beta \in \Gamma^*$, and $\alpha \in \overline{\Sigma}^*$.

In other words, condition (i) is a shift-
read-move) while condition (ii) is a reduce-mo
stack-update move). Note that a shift-move ca
read-head to be advanced one symbol on the inp
and the symbol A is pushed onto the stack. On
other hand, the reduce-move leaves the read-he
input tape stationary, removes (pops) the stri
the stack $\beta B\theta$, and adds (pushes) the symbol A
new top of stack resulting in the stack βBA.
Furthermore, after removing the string θ from
stack, the new top of the stack is consulted t
that it is labeled with the symbol B (i.e. pe
stack look-back of one symbol).

Acceptance, in a computation, occurs if t
computation reaches an instantaneous descripti
the end of the input string is reached and has
stack. That is, the <u>language accepted by a PD</u>
denoted N(D), is the set

$$N(D) = \{\alpha \in \overline{\Sigma}^* \mid (\underline{|}, \alpha) \vdash^* (\underline{|}, \varepsilon)\}$$

where $\vdash^*$ is the transitive reflexive closure o

DA is considered deterministic if and only if
ry instantaneous description id_1, if there is an
aneous description id_2 such that $id_1 \vdash id_2$,
$_2$ is unique. That is, a PDA is <u>deterministic</u> if
y if either

for all $a \in \Sigma$ and $B \in \Gamma$, $|\delta(B,a)| \leq 1$

for all $B \in \Gamma$ and $\alpha \in \Gamma^*$, $|\delta(B\alpha,\varepsilon)| \leq 1$.
Furthermore, if $\delta(B\alpha,\varepsilon)$ is a singleton set,
then for any string $\theta_1\theta_2 = B\alpha$ where $\theta_1 \neq \varepsilon$,
$|\delta(\theta_2,\varepsilon)| = 0$.

for all $a \in \Sigma$, $B \in \Gamma$, and $\alpha \in \Gamma^*$, if $|\delta(B,a)| = 1$ the
$\delta(\alpha B,\varepsilon) = \emptyset$ and if $|\delta(\alpha B,\varepsilon)| = 1$ then $\delta(B,a) = \emptyset$

, condition (i) guarantees that there is only
sible shift move that is applicable (i.e.
ees that there can not be a shift/shift
t), condition (ii) guarantees that there is only
sible reduce move that is applicable (i.e.
ees that there can not be a reduce/reduce
t), and condition (iii) guarantees that if a
ove is applicable, then there is not a reduce
at is applicable and vice versa (i.e.
ees that there can not be a shift/reduce
t).

Example 3.3.1: Let $D_1=(\Sigma,\Gamma,\delta,\underline{|})$ be a PDA

Σ = {a,b};

Γ = $\{\underline{|}$,a,b,A,S}; and

δ is defined by the following table w

input pair (α,β), the rows represent

the columns represent values of β. F

empty table entries represent null se

	a	b	ε
$\underline{\vert}$	{a}		$\{\underline{\vert}A\}$
a	{a}		{aA}
A		{b}	
aaAb			{aA}
$\underline{\vert}$aAb			$\{\underline{\vert}A\}$
$\underline{\vert}$A			$\{\underline{\vert}S\}$
$\underline{\vert}$S			$\{\underline{\vert}\}$

The language accepted by D_1 is the set

$N(D_1) = \{a^n b^n \mid n \geq 0\}$. For example, the

is accepted as follows:

$(\underline{|},aaabbb) \vdash (\underline{|}a,aabbb) \vdash$

$(\underline{|}aa,abbb) \vdash (\underline{|}aaa,bbb) \vdash$

$(\underline{|}aaaA,bbb) \vdash (\underline{|}aaaAb,bb) \vdash$

$(\underline{|}aaA,bb) \vdash (\underline{|}aaAb,b) \vdash$

$(\underline{|}aA,b) \vdash (\underline{|}aAb,\varepsilon) \vdash$

$(\underline{|}A,\varepsilon) \vdash (\underline{|}S,\varepsilon) \vdash (\underline{|},\varepsilon)$

h is an accepting condition. Also, D_1 is not rministic since there is a shift/reduce conflict een $\delta(\perp,a)$ and $\delta(\perp,\epsilon)$.

The main result about PDAs used by LR(0) parsers hat the class of languages accepted by eterministic PDAs is precisely the class of ext-free languages which is stated by the followi rem:

rem 3.3.1: (Chomsky[62], Schutzenberger[63], or [63]) The class of string languages generated by ng grammars is identical to the class of string uages accepted by PDAs.

LR(0) Parsers

An LR(0) parser is a PDA which is presented in a htly different format. That is, the transition tion δ is implicitly defined by a set of parsing es generated from some given string grammar. In , the LR(0) parsing tables are a "compressed" esentation of the transition map δ. Hence, a LR(er can be viewed as consisting of two parts, a er routine and a set of parsing tables generated the string grammar given (see figure 3.4.1).

```
           +-------------------------
      +-->| I n p u t   t a p e ...
      |    +-------------------------
      |
+-------------+      +----------+
| Driver      |      | Parsing  |
|             |----->|          |
|     Routine |      | tables   |
+-------------+      +----------+
            |
+---------------+   |
|Stack of symbols|<--+
+---------------+
```

Figure 3.4.1: Organization of an LR(0) p

In generating the LR(0) parser, from a g string grammar G, the tables are built so tha parser traces a rightmost derivation in rever is accomplished by essentially scanning the s bottom to top, between each move, to determin sentential forms, if any, could exist with th prefix defined by the stack. It turns out th can be done using a FSA (called the character automaton) which parses the stack to recogniz string prefix the stack matches. Furthermore not necessary to read the stack from bottom t every move. Rather, by encoding the elements stack to uniquely determine both the string p parsed and the states of the characteristic a

top of the stack will always be a symbol which
ntifies what state the characteristic automaton
ld be in if the stack was scanned from bottom to
. Thus, the LR(0) parser can determine all the
ormation it needs to know by only inspecting the
the stack and the next input symbol.

This section begins by presenting the LR(0) par
terms of its parsing tables. It continues in
tion 3.4.2 by presenting how the LR(0)
racteristic automaton is built. Section 3.4.3
sents how the LR(0) parser is generated from the
racteristic automaton. Finally, section 3.4.4
cludes this section by presenting how a LR(0) par
be transformed into a PDA as defined in section
.

.1 LR(0) Parsing Tables -

The LR(0) parser is a machine which has string
ut, uses a stack, and three parsing tables. The
ck is a string of "states" which implicitly holds
ormation on both the string of grammar symbols
ognized and the states of the characteristic
omaton used to parse the stack. More formally, a
0) parser M is a 6-tuple M=(G, K, <u>shift</u>, <u>reduce</u>,

to, <u>start</u>) where

G = (Φ,Σ,P,S) is the string grammar defining the LR(0) parser;

K is a finite set of parser <u>states</u>;

<u>shift</u> : K x Σ -> K$\vee${<u>error</u>} is a function defini the parsing <u>shift</u> <u>table</u>;

<u>reduce</u> : K -> 2^P is a function defining the parsing <u>reduce</u> <u>table</u>;

<u>goto</u> : K x Φ -> K$\vee${<u>error</u>} is a function defining the parsing <u>goto</u> <u>table</u>; and

<u>start</u>$\in$K is the <u>initial</u> <u>state</u> and defines the empty stack symbol.

An LR(0) parser is considered <u>well</u> <u>defined</u> if ly if the LR(0) parser is deterministic (i.e. no ving any shift/reduce or reduce/reduce conflicts) other words, an LR(0) parser is well defined if ly if

i) for all k$\in$K, $|$<u>reduce</u>(k)$| \leq 1$

ii) for all k$\in$K, for all a$\in\Sigma$, if <u>shift</u>(k,a)$\in$K, then <u>reduce</u>(k)=$\emptyset$.

As stated earlier, an LR(O) parser is just a rent presentation of a deterministic PDA. Hence, stantaneous description of an LR(0) parser ted ID) is the same as for a PDA. That is, an ntaneous descriptions is a pair (j3,oc)€K x *X* p is the current stack and oc is the string left an on the input tape. The initial configuration e pair (start, oc) where oc is the string to parse

The decision relation k $_a$: ID x ID of an LR(0) r M*(G»(l,i,P,S), K, shift, reduce, goto, start) mines the next move made by the LR(0) parser M. is, given two instantaneous descriptions id_1 and ±d_l h_d id_2 if and only if

i) id^CjJk^aoc) and id_2»(j i k ^ , oc) where

shift$(k_1,a)=k_2$

i) id^CjiqQ, oc) and id^Cpqgqj^ , oc) where

reduce(q_Q)={A->6} and goto(q_Q,A)«q_t

i) id^CpqQkj . . .k_n, oc) and *Id^i^q^q^₉ cc)* where

n>l; $k_1,...,k_n$€K, reduce(k_n)~{A->8},

length(e)=«n, and goto$(q_Q, A)=^t q_1$

iv) id_1=(<u>start</u>q_0,ε) and id_2=(<u>start</u>,ε) where <u>reduce</u>(q_0)={S->S'}.

other words, condition (i) is a shift-move, ndition (ii) is a reduce-move on an epsilon rule, ndition (iii) is a reduce-move on a non-epsilon r d condition (iv) is a reduce-move on the start oduction causing acceptance.

Acceptance of a string α only occurs if the cision relation reaches an instantaneous descript the form (<u>start</u>,ε). That is, the <u>language accep</u> <u>a well defined LR(0) parser M</u>, denoted N(M), is t

$$N(M) = \{\alpha \in \Sigma^* \mid (\underline{start},\alpha) \vdash_d^* (\underline{start},\varepsilon)\}$$

ere $\vdash_d^*$ is the transitive reflexive closure of $\vdash_d$.

ample 3.4.1: Let M = (G,K,<u>shift</u>,<u>reduce</u>,<u>goto</u>,1) wh

G = (Φ,Σ,P,S) such that

Φ = {S,A};

Σ = {a,b}; and

P = {S->A, A->ab, A->aAb};

K = {1,2,3,4,5,6}; and

<u>shift</u>,<u>reduce</u> and <u>goto</u> are defined by the followi tables:

shift

	a	b
1	3	
3	3	4
5		6

reduce

2	S->A
4	A->ab
6	A->aAb

goto

	A
1	2
3	5

language accepted by the LR(0) parser M is the se

$(M) = \{a^n b^n \mid n \geq 1\}$

example, the string "aaabbb" is accepted as

ows:

$1,aaabbb) \vdash_d (13,aabbb) \vdash_d (133,abbb)$

$\vdash_d (1333,bbb) \vdash_d (13334,bb) \vdash_d (1335,bb)$

$\vdash_d (13356,b) \vdash_d (135,b) \vdash_d (1356,\varepsilon)$

$\vdash_d (12,\varepsilon) \vdash_d (1,\varepsilon)$ which is the accepting conditi

One should note that there is a relationship een the set of states on the stack of the LR(0) er and the corresponding grammar that the LR(0) er is based on. This relationship is known as th lling" as is defined as follows:

Given an LR(0) parser $M=(G=(\Phi,\Sigma,P,S)$, K, <u>shif</u> <u>educe</u>, <u>goto</u>, <u>start</u>), let spelling : $K^* \rightarrow 2^{(\Phi \vee \Sigma)^*}$ e a function recursively defined such that for ar tring of states $s_1 s_2 \ldots s_n \in K^n$

i) spelling(ε) = spelling(<u>start</u>) = $\emptyset$

ii) spelling($s_1 \ldots s_n$) = $\emptyset$ if $n \geq 2$ and either

a) there does not exist a symbol $a \in \Sigma$ such shift(s_1,a)=s_2 and there does not exist symbol $A \in \Phi$ such that goto(s_1,A)=s_2

b) spelling($s_2 \ldots s_n$) = $\emptyset$

iii) spelling($s_1 \ldots s_n$) = $\{a\beta \mid \beta \in$ spelling($s_2 \ldots s$ $a \in \Sigma$, and $s_2 \in$ shift(s_1,a)$\} \vee$
$\{A\beta \mid \beta \in$ spelling($s_2 \ldots s_n$), $A \in \Phi$, and $s_2 \in$ goto(s_1,A)$\}$ otherwise

.4.2 **LR(0) Characteristic Automaton -**

An LR(0) parser M is constructed based on a g tring grammar G=(Φ,Σ,P,S). In generating M, the onstruction method tries to maintain the property hat for any input string $\theta\alpha$, if $(\varepsilon,\theta\alpha) \vdash_d^* (\beta,\alpha$ ne of the following two conditions hold:

i) if $\theta\alpha$ is a string generated by G, then

$$S \overset{*}{\underset{R}{\Longrightarrow}} \beta\alpha \overset{*}{\underset{R}{\Longrightarrow}} \theta\alpha$$

i) there exists a string $\alpha' \in \Sigma^*$ such that

$$S \overset{*}{\underset{R}{\Rightarrow}} \beta\alpha' \overset{*}{\underset{R}{\Rightarrow}} \theta\alpha'$$

ote: While in a LR(0) parser the stack is a stri
f states, this discussion assumes that the stack
eferenced corresponds to the "spelling" of the
tring of states which is a string of grammar
ymbols.

ther words, the construction method tries to
tain the property that every instantaneous
ription corresponds to some legal sentential form
ition (i) states that this will be the case
ever the input string is legal while condition (i
es that even if the input is illegal, there exist
ring $\theta\alpha' \in L(G)$ such that if $\theta\alpha'$ was the input, t
instantaneous description would correspond to a
l sentential form (i.e. the scanned input string
a legal suffix α' such that $\theta\alpha' \in L(G)$).

Another way of looking at the above condition is
the construction method for the LR(0) parser wil
d a PDA where every reduce-move will be defined t
orm the inverse of some derivation step and the
rse of every derivation step will be defined by
reduce-move. Hence, for any sentential form $\alpha\beta$
e $\alpha\beta \in (\Phi \vee \Sigma)^*$, $\theta \in \Sigma^*$, and any production $A \rightarrow \beta$ in P,

if the current instantaneous description is the p

$(\alpha\beta,\theta)$, one wants to create a reduce-move such t

$(\alpha\beta,\theta) \vdash_d (\alpha A,\theta)$. To accomplish this, one must

way of recognizing all possible stack configurati

which a reduce-move should be defined (i.e. when

reverse of a derivation step should be performed)

Clearly, from above, the set of all such stack

configurations is the set $\{\alpha\beta \mid S \overset{*}{\underset{R}{\Longrightarrow}} \alpha A\theta \underset{R}{\Longrightarrow} \alpha\beta\theta\}$

string $\alpha\beta$ in this set is called a <u>characteristic</u>

<u>string</u>. Let CS_G denote the set of all characteri

strings. That is, given a string grammar G,

$CS_G = \{\alpha\beta \mid S \overset{*}{\underset{R}{\Longrightarrow}} \alpha A\theta \underset{R}{\Longrightarrow} \alpha\beta\theta\}$.

It is an important result that given a strin

grammar $G=(\Phi,\Sigma,P,S)$, the set of characteristic st

CS_G is generated by a strict right-linear string

grammar (see Knuth[68]). The method used, for

constructing the strict right-linear grammar G_C i

create a new set of nonterminals, where each

nonterminal is a "marked production". That is, a

<u>marked production</u> is a pair $(A\text{->}\alpha, i)$ where $A\text{->}\alpha$

production in P and $0 \leq i \leq \text{length}(\alpha)$ is a marker wi

the production $A\text{->}\alpha$. A marked production $(A\text{->}\alpha$

be denoted as $(A\text{->}\beta_1 . \beta_2)$ where $. \notin \Sigma \vee \Phi$, $\alpha=\beta_1\beta_2$, ar

$i=\text{length}(\beta_1)$. Also, let mp(P) denote the set of

marked productions defined by the set of producti

Using marked productions, the conversion can easily be defined as follows:

Definition 3.4.1: (Geller and Harrison[77], Harrison[78]) Given the string grammar $G=(\overline{\Phi},\overline{\Sigma},P,S$ its corresponding characteristic grammar C_G be th string grammar $C_G=(mp(P)\vee S',\overline{\Phi}\vee\overline{\Sigma},P',S')$ where P' the set of productions defined as follows:

i) for all $S\text{->}\alpha\in P$, $S'\text{->}(S\text{->}.\alpha)\in P'$

ii) for any $A\in\overline{\Phi}$, $\underline{X}\in\overline{\Phi}\vee\overline{\Sigma}$, and $\alpha,\beta\in(\overline{\Phi}\vee\overline{\Sigma})^*$ suc $(A\text{->}\alpha.\underline{X}\beta)\in mp(P)$, $(A\text{->}\alpha.\underline{X}\beta)\text{->}\underline{X}(A\text{->}\alpha\underline{X}.\beta$

iii) for any $A,B\in\overline{\Phi}$, $B\text{->}\beta\in P$, and $\alpha,\theta\in(\overline{\Phi}\vee\overline{\Sigma})^*$ s that $(A\text{->}\alpha.B\theta)\in mp(P)$, $(A\text{->}\alpha B\theta)\text{->}(B\text{->}.\beta$

iv) For any $A\in\overline{\Phi}$ and $\alpha\in(\overline{\Phi}\vee\overline{\Sigma})^*$ such that $(A\text{->}\alpha.)\in mp(P)$, $(A\text{->}\alpha.)\text{->}\varepsilon$

Theorem 3.4.1: Given any string grammar G, and it corresponding characteristic grammar C_G as define definition 3.4.1, $L(C_G)=CS_G$.

Example 3.4.1: Let $G=(\overline{\Phi},\overline{\Sigma},P,S)$ be a string gramma where

$\overline{\Phi}$ = {S,A};

$\overline{\Sigma}$ = {a,b}; and

$P = \{S \to A,\ A \to \varepsilon,\ A \to aAb\}$.

Clearly, $L(G) = \{a^n b^n \mid n \geq 0\}$ and the set of characteristic strings is the set

$CS_G = \{\varepsilon, A\} \cup \{a^n Ab \mid n \geq 1\}$. Furthermore, the right-linear grammar C_G defined by definition the string grammar $C_G = (mp(P) \cup \{S'\}, \Phi \cup \Sigma, P', S')$ contains the productions

$$
\begin{array}{l}
S' \to (S \to .A) \\
(S \to .A) \to (A \to .\varepsilon) \\
(A \to .\varepsilon) \to \varepsilon \\
(S \to .A) \to (A \to .aAb) \\
(S \to .A) \to A(S \to A.) \\
(S \to A>) \to \varepsilon \\
(A \to .aAb) \to a(A \to a.Ab) \\
(A \to a.Ab) \to (A \to .aAb) \\
(A \to a.Ab) \to (A \to .\varepsilon) \\
(A \to a.Ab) \to A(A \to aA.b) \\
(A \to aA.b) \to b(A \to aAb.) \\
(A \to aAb.) \to \varepsilon
\end{array}
$$

For example, a derivation in G is

$$S \underset{R}{\Longrightarrow} A \underset{R}{\Longrightarrow} aAb \underset{R}{\Longrightarrow} aaAbb \underset{R}{\Longrightarrow} aaaAbbb$$

and hence "aaaAb" is a characteristic string. corresponding derivation in C_G, which generat characteristic string "aaaAb", is as follows:

$$
\begin{array}{l}
S' \underset{R}{\Longrightarrow} (S \to .A) \underset{R}{\Longrightarrow} (A \to .aAb) \underset{R}{\Longrightarrow} \\
a(A \to a.Ab) \underset{R}{\Longrightarrow} a(A \to .aAb) \underset{R}{\Longrightarrow} \\
aa(A \to a.Ab) \underset{R}{\Longrightarrow} aa(A \to .aAb) \underset{R}{\Longrightarrow} \\
aaa(A \to a.Ab) \underset{R}{\Longrightarrow} aaaA(A \to aA.b) \underset{R}{\Longrightarrow} \\
aaaAb(A \to aAb.) \underset{R}{\Longrightarrow} aaaAb
\end{array}
$$

Using the results of theorem 3.2.4, one ca
the characteristic grammar C_G and create a
deterministic FSA CG to accept the set of
characteristic strings. However, rather than g
through the three different conversions separat
(i.e. construction the string grammar C_G, buil
nondeterministic FSA M from C_G, and building th
deterministic FSA CG from M), these three conve
can be combined into a single algorithm as foll

Algorithm 3.4.1: Method to construct an LR(0) characteristic automaton

Input: a string grammar $G=(\Phi,\Sigma,P,S)$

Output: a Deterministic FSA $CG=(\Sigma,K,\delta,q_0,F)$ without epsilon rules.

Method: The three procedures below, initiated by calling ITEMS(G);

<u>Procedure</u> ITEMS(G);

   <u>begin</u>

      <u>For</u> all input pairs $(a,b) \in K \times (\Sigma \vee \{\varepsilon\})$,

         let $\delta(a,b)=\emptyset$;

      q_0 := closure($\{(S\text{->}.\alpha) \mid S\text{->}\alpha \in P\}$);

      K := $\{q_0\}$;

```
        repeat
            for each set of marked productic
                and each grammar symbol X€(§\
                such that J=GOTO(I,X) and Jj
            do
                K :« K V {J};
                &(I,X) := {J};
            od;
        until no more sets of marked produc
              can be added to K;
        F := 0;
        For  each I€K do
            if there exists a marked product
                of the form (A->oc.)GI
                then F := F V {1}
            it?
        od;
    end;
Function GOTO(I,X);
    begin
        J := {(A->ocX*p) I (A->oc .Xβ)€I}
        return closure(J);
    end
```

```
tion closure(I);
egin
   J := I;
   while there exists a marked production
        of the form (A->α.Bβ)∈I such that
        B->θ∈P and (B->.θ)∉J
   do
      J := J ∨ {(B->.θ)};
   od;
   return J;
nd;
```

mple 3.4.2: Let $G=(\Phi,\Sigma,P,S)$ be a string grammar
re

Φ = {S,A};

Σ = {a,b}; and

P = {S->A, A->ab, A->aAb}.

n, the deterministic FSA CG to accept the set of racteristic string CS_G is the FSA $CG=(\Sigma,K,\delta,q_0,Q)$
re

Σ = {a,b};

K = {{(S->.A), (A->.ab), (A->.aAb)},
    {(S->A.)},
    {(A->a.b), (A->a.Ab), (A->.ab), A->.aAb)},
    {(A->ab.)},
    {(A->aA.b)},

{(A->aAb.)}};

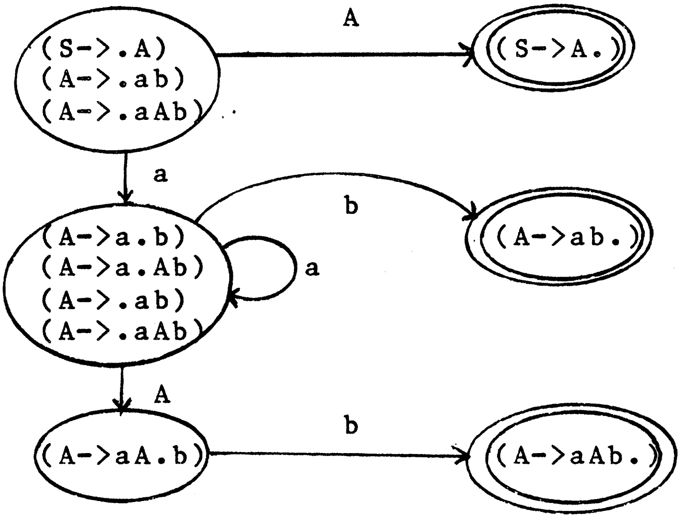
q_0 = {(S->.A),(A->.ab),(A->.aAb)};

Q = {{(S->A.)},

{(A->aAb.)},

{(A->ab.)}}; and

δ is defined by the following graph:



3.4.3 Constructing LR(0) Parsing Tables -

This section presents how to construct an parser from the characteristic automaton. The construction does not always construct a well d LR(0) parser (i.e. it may produce a nondetermi PDA). However, for a subset of the string gram known as "LR(0) grammars", it is guaranteed to a well defined LR(0) parser.

The process of conversion is straightforward
resented by the following Algorithm:

lgorithm 3.4.2: Constructing an LR(0) parser

nput: a string grammar $G=(\Phi,\Sigma,P,S)$ and its corresponding characteristic automaton $CG=(\Sigma,K,\delta,q_0,F)$

utput: an LR(0) parser $M=(G,C,$<u>shift</u>,<u>reduce</u>,<u>goto</u>,<u>start</u>$)$

ethod: Let $K=\{I_1,I_2,...,I_n\}$ be the set of sets of
arked productions from the characteristic automat
G. Then $C=\{1,2,...,n\}$ where state i corresponds
he set of marked productions I_i. let <u>start</u>=k whe
$_k=q_0$ is the start state of the characteristic
utomaton CG. The three parsing tables are define
ollows:

hift table:

For all $i\in C$, all productions $A\to\alpha\in P$, and all a
$(A\to\beta.a\theta)\in I_i$ where $\beta a\theta=\alpha$ and $I_j\in\delta(I_i,a)$, then
<u>shift</u>(i,a)=j. Otherwise <u>shift</u>(i,a)=<u>error</u>.

educe table:

For all states $i\in C$, <u>reduce</u>$(i)=\{A\to\alpha \mid (A\to\alpha.)$

goto table:

For all $i \in C$, all productions $A \to \alpha \in P$, and al

$A \to \beta . B\theta \in I_i$ where $\beta B \theta = \alpha$ and $I_j \in \delta(I_i, B)$, then

goto(i,B)=j. Otherwise goto(i,a)=error.

Example 3.4.3: Let $G=(\Phi,\Sigma,P,S)$ and $CG=(\Sigma,K,\delta,q_0$

defined in example 3.4.2. Then, using algorith

the constructed LR(0) parser is

M=(G,C,shift,reduce,goto,1) where

C={1,2,3,4,5,6} such that

I_1={(S->.A), (A->.ab), (A->.aAb)},

I_2={(S->A.)},

I_3={(A->a.b), (A->a.Ab), (A->.ab), (A->.a

I_4={(A->ab.)},

I_5={(A->aA.b)}, and

I_6={(A->aAb.)};

and the three parsing tables are defined as

shift

	a	b
1	3	
3	3	4
5		6

reduce

2	S->A
4	A->ab
6	A->aAb

goto

	A
1	2
3	5

Note that this corresponds to the LR(0) pars

presented in example 3.4.1.

As mentioned earlier, the above algorithm does
essarily guarantee to produce a well defined LR(0
ser. Rather, it only guarantees a well defined
0) parser if the given string grammar is an <u>LR(0)</u>
<u>mmar</u>. That is, a string grammar $G=(\Phi,\Sigma,P,S)$ is
0) if and only if for any two derivations $S \overset{*}{\underset{R}{\Longrightarrow}}$
$\alpha\beta\theta$ and $S \overset{*}{\underset{R}{\Longrightarrow}} \alpha'A'\theta' \underset{R}{\Longrightarrow} \alpha'\beta'\theta'$, if $\alpha\beta$ is a
$\alpha'\beta'$, then $\alpha=\alpha'$, $\beta=\beta'$, and $A=A'$.

.4 Converting LR Parsers To PDAs -

As mentioned throughout this section, an LR(0)
ser is nothing more than a PDA while a well defi
0) parser is a deterministic PDA. To show this,
s section provides an algorithm which will conve
0) parser into a PDA.

The conversion is done by constructing the
nsition map δ such that the computation relation
ulates the decision function $\vdash_d$. The algorithm
omplish this is as follows:

Algorithm 3.4.3: Converting an LR(0) parser into a PDA

input: An LR(0) parser $M=(G=(\Phi,\Sigma,P,S),K,\underline{shift},\underline{reduce},\underline{goto},\underline{start})$ a "spelling" function of its corresponding characteristic automaton.

output: a PDA $D=(\Sigma,K,\delta,\underline{start})$

Method: The procedure below which constructs the function δ.

```
procedure convert(M,spelling);
  begin
    for all input pairs (a,b) do
      initialize δ(a,b)=∅.
    od;
```

```
for all states k∈K do
    for all a∈Σ do
        if shift(i,a)=j
            then δ(i,a)={j}
        fi
    od
    for all productions A->ε∈reduce(k) do
        for all q such that goto(k,A)=q do
            δ(k,ε) := δ(k,ε)∨{kq}
        od
    od
    for all productions A->α∈reduce(k)
        such that A≠S and length(α)≥1 do
            for all q1,q2∈K such that goto(q1,A)=q
            do
                for all βk∈K^length(α) such that
                spelling(βk)=α do
                    δ(q1βk,ε) := δ(q1βk,ε) ∨ {q1q2}
                od
            od
    od
```

```
      for all productions S->S'∈reduce(k) do
         δ(startk,ε) := δ(startk,ε) V {start}
      od
   od
end
```

xample 3.4.4: Let M=(G,C,shift,reduce,goto,1) be R(0) parser defined in example 3.4.3. Then the orresponding PDA D=(Σ,C,δ,1), defined by algorith .4.3, has its transition function defined by the ollowing table:

	a	b	ε
1	3		
12			1
3	3	4	
134			12
334			35
5			
1356			12
3356			35

Chapter IV

CONTEXT-FREE TREE LANGUAGES

A tree language is simply some subset of the set 1 finite trees (i.e. T_Σ for the ranked alphabet Except for trivial cases, a tree language is an ite set. Even though such sets may be infinite, ould like to have finite means for defining them, uch method is a generative device called a ar. A grammar is a set of rules which defines tree in a tree language. Of interest here is the of tree languages which are generated by xt-free tree grammars, called the class of xt-free tree languages.

This chapter begins by presenting the defi
of a context-free tree grammar, and by showing
tree language is generated from a given context-
tree grammar via a series of derivation (or rew
steps. This includes presenting the two types
restricted forms of derivations, known as outsi
and inside-out. The remainder of the chapter s
properties of context-free tree grammars, as we
transformations on some of these grammars which
the grammar such that certain undesired propert
removed. One goal is to provide a standard for
context-free tree grammars which are said to be
Chomsky normal form. Another goal is to descri
distinct forms of tree grammars, known as root
linear tree grammars, which generate distinct
subclasses of tree languages called regular and
coregular tree languages.

4.1 Context-free Tree Grammars And Tree Langua

This section defines the set of trees (or
language) generated by a tree grammar. The gen
process can be characterized as a series of suc
tree rewrites (or one-step derivations) until a
which is only labeled with terminal symbols is
generated. This section provides the definitio

ree grammar, followed by an overview of the gene

rocess.

A context-free tree grammar (or tree grammar

hort) is a quadruple (Φ,Σ,P,F_1) where

$\Phi = \{F_1,F_2,\ldots,F_n\}$ is a finite ranked alphabet
nonterminal function symbols (where the ari
each F_i, $1\le i\le n$, is denoted as a_i),

$\Sigma = \{f_1,f_2,\ldots,f_m\}$ is a finite ranked alphabet
terminal function symbols,

F_1 is a designated symbol in Φ called the
start symbol and

P is a finite set of pairs of trees of the for

$$\left(\begin{array}{c} F_i \\ / \;\; \backslash \\ x_1 \ldots x_{a_i} \end{array} ,\; t\right),$$

for any i, $1\le i\le n$, and t is a finite tree in
$T_{\Sigma\vee\Phi}(X_{a_i})$.

Each pair $\left(\begin{array}{c} F_i \\ / \;\; \backslash \\ x_1 \ldots x_{a_i} \end{array} ,\; t\right) \in P$ is called a production

Note that under tree composition ,

$$\begin{array}{c} g \\ / \;\; \backslash \\ x_1 \ldots x_n \end{array} = g(x_1,\ldots,x_n)$$

ere $g \in \Sigma \vee \Phi$ and $n=r(g)$. For convenience of notatio $(x_1,\ldots,x_n)$ will be denoted in vector form as $g(\vec{x})$ oductions will be denoted as $F_i(\vec{x}) \to t$ where $(\vec{x}),t) \in P$. In general, upper case letters such a G,H,... will be used to denote nonterminal symbo ile lower case letters such as f,g,h,... will be ed to denote terminal symbols. Depending on the ntext, G will also be used to denote a tree gramm rthermore, unless otherwise defined, one can assu at $A = \max\{a_i \mid a_i = r(F_i) \text{ and } F_i \in \Phi\}$.

Having defined a tree grammar, the next step i fine a rewrite step. Given a tree grammar (Φ,Σ,P,F_1), a <u>one-step derivation</u> (or rewrite) is fined by the relation $\underset{u}{\Longrightarrow}_G \subseteq T_{\Sigma\vee\Phi}(X_A) \times T_{\Sigma\vee\Phi}(X_A)$ llows:

For any two trees $t_1,t_2 \in T_{\Sigma\vee\Phi}(X_A)$, $t_1 \underset{u}{\Longrightarrow}_G t_2$ if only if $t_1 = s[v \leftarrow F(s_1,\ldots,s_q)]$ and $t_2 = s[v \leftarrow t(s_1,\ldots,s_q)]$ where $r(F)=q$, $s,s_1,\ldots,s_q \in T_{\Sigma\vee\Phi}(X_A)$, $v \in dom(s)$, and $F(\vec{x}) \to t$ is a production in P.

other words, the subtree $F(s_1,\ldots,s_q)$, of the tr , is rewritten (or replaced) with the tree $(s_1,\ldots,s_q)$ using the production $F(\vec{x}) \to t$. When th ntext of G is clearly known, $\underset{u}{\Longrightarrow}_G$ will simply be noted as $\Longrightarrow$.

Equipped with the meaning of a one-step erivation, one is able to define precisely the se rees generated from a tree grammar. Let $G=(\bar{\Phi},\bar{\Sigma},P$ a tree grammar and $b_1,\ldots,b_{a_1}$ be a sequence of erminal trees in $T_{\bar{\Sigma}}$. A <u>sentential</u> <u>form</u> is any tr $\in T_{\bar{\Sigma}\vee\bar{\Phi}}$ such that $F_1(b_1,\ldots,b_{a_1}) \Longrightarrow^* t$ where $\Longrightarrow^*$ ne transitive reflexive closure of $\Longrightarrow$. Furtherm ne <u>tree</u> <u>language</u> <u>generated</u> <u>by</u> <u>the</u> <u>tree</u> <u>grammar</u> <u>G</u>, enoted $L(G,F_1(b_1,\ldots,b_{a_1}))$, is the set of all entential forms t such that $t\in T_{\bar{\Sigma}}$. Hence,

$(G,F_1(b_1,\ldots,b_{a_1})) = \{t \mid F_1(b_1,\ldots,b_{a_1}) \Longrightarrow^* t$ a $\in T_{\bar{\Sigma}}\}$.

<u>**xample**</u> <u>**4.1.1:**</u> Let $G_1=(\bar{\Phi},\bar{\Sigma},P,F)$ such that

$\bar{\Phi}$ = {F} where r(F)=1,

$\bar{\Sigma}$ = {f,a} where r(f)=1 and r(a)=0, and

```
P = { F -> x , F -> f}.
      |        |    |
      x.       x    F
                    |
                    x
```

```
Then, L(G,F(a)) = {a, f, f, f, ... }.
                      |  |  |
                      a  f  f
                         |  |
                         a  f
                            |
                            a
```

Also, a sample derivation which derives

```
f
|
f
|
a
```

is as follows:

```
F ==> f ==> f ==> f
|     |     |     |
a     F     f     f
      |     |     |
      a     F     a
            |
            a
```

It should be noted that the situation regar derivations is not as simple as in the case of s grammars. Unlike in string grammars, one-step derivations are not commutative in the sense tha $t_1 \Longrightarrow t_2$ using $F(\vec{x})\text{->}s_1$, and $t_2 \Longrightarrow t_3$ using G(it is not necessarily the case that there exists t_2' such that $t_1 \Longrightarrow t_2'$ using $G(\vec{x})\text{->}s_2$ and $t_2' \Longrightarrow$ using $F(\vec{x})\text{->}s_1$. To show this, consider the foll example:

ample **4.1.2:** Let $G_2=(\Phi,\Sigma,P,S)$ be a tree grammar s
at:

Φ = {S,F,G} where r(S)=0 and r(F)=r(G)=1;

Σ = {a,g} where r(a)=0 and r(g)=1; and

```
P = {S->F , F->a , G->g}
        |    |      |   |
        G    x      x   x
        |
        a
```

```
Clearly, F ==> F using G->g and F ==> a using F-
         |     |        |  |      |            |
         G     g        x  x      g            x
         |     |                  |
         a     a                  a
```

On the other hand, when the order of the derivat
steps is reversed,

```
F ==> a using F->a and it is now
|             |
G             x
|
a
```

```
impossible to perform a rewrite using G->g.
                                      |  |
                                      x  x
```

Hence, the order in which derivation steps are
plied affects the resulting derived tree (i.e.
rivation steps are not independent of one anothe
is has been shown by Englefriedt and Schmidt[77,7
rthermore, this result indicates that any proof
owing results between two different derivations m

consider every possible ordering of derivation
To simplify proofs, one would like to have a m
derivation (i.e. prior knowledge about actual
orderings of derivations that will occur) whic
reduce the number of potential derivation orde
must consider, even at the cost of restricting
class of tree languages allowed. Thus, one sh
consider what modes of derivation one will all
least restrictive is not to specify any deriva
(i.e. unrestricted as above). However, the c
practice is to put a partial ordering on the d
steps by using either an outside-in (OI) or an
inside-out (IO) derivation mode. Intuitively,
modes of derivation correspond to call by name
by value respectively.

An <u>IO</u> <u>one-step</u> <u>derivation</u> (denoted $\underset{IO}{\Longrightarrow}_G$)
one-step derivation which is applied to an inn
nonterminal occurring in a subtree. It can be
to <u>any</u> subtree whose root is labeled with a no
symbol and none of its other nodes are labeled
nonterminal symbol. Note that an IO derivatio
applied to any subtree meeting the above condi
More formally, the $\underset{IO}{\Longrightarrow}_G$ relation is defined as

For any two trees $t_1, t_2 \in T_{\Sigma \vee \Phi}(X_A)$, $t_1 \underset{IO}{\Longrightarrow}_G$ t
only if $t_1 \Longrightarrow t_2$ such that

i) $t_1 = s[u \leftarrow F(s_1,\ldots,s_n)]$

ii) $t_2 = s[u \leftarrow t(s_1,\ldots,s_n)]$

iii) $F(\vec{x}) \rightarrow t$ is a production in P where $r(F)=n$, an

iv) for all $v \in N_+^+$ such that $u \cdot v \in dom(t_1)$, where N_+^+ denotes the set of nonempty strings of positi integers, $t_1(u \cdot v) \notin \bar{\Phi}$.

e that conditions i) through iii) are just the iitions of a one-step derivation while condition the added condition for an IO derivation.

Similarly, an <u>OI</u> <u>one-step</u> <u>derivation</u> (denoted $_G$) is a one-step derivation applied to an outermo terminal symbol. It can be applied to <u>any</u> node eled with a nonterminal symbol such that none of estor nodes are labeled with a nonterminal symbol e an IO derivation, an OI one-step derivation may lied to any subtree meeting the above condition. e formally, the relation $\overline{\overline{OI}}\!\!>_G$ is defined as follo For any two trees $t_1, t_2 \in T_{\Sigma \vee \bar{\Phi}}(X_A)$, $t_1 \overline{\overline{OI}}\!\!>_G t_2$ if a only if $t_1 \Longrightarrow t_2$ such that

i) t_1 = s[u <- F(s_1,...,s_n)]

ii) t_2 - s[u <- tCsj,...,s_n)]

iii) F(*)->t is a production in P where r(F)

iv) for all prefixes v of u, when v^u, t_1(v

Again, as in an 10 one-step derivation, condit through iii) are just the conditions for a one derivation while condition iv) is the added co for an 01 derivation. Furthermore, whenever G fixed, «> „ and *Ir5>*$_n$ will simply be denoted as $\overset{==>}{OI}$ respectively.

To clarify the differences between unrest 10, and 01 derivations (i.e. =>, $\underset{IO}{===>}$, and $\underset{OI}{==}$ consider the following example:

Example 4.1*3: Let G_3 = (|,Ī, P,S) be a tree gram that

1 » {S,F,G} where r(S)»0 and r(F)-r(G)-l;

2 « {f,g,a} where r(f)«r(g)»2 and r(a)«l;

```
P - {S->F , F->f , G->g , G->x}
        I   I / \  I / \  I
        G   x  x  x  x  x  x  x
        I
        a
```

The set of all possible 10 derivations is a follows:

```
S ==> F ==> F ==>     f
  IO  |  IO | IO     / \
      G     g      g     g
      |    / \    / \   / \
      a   a   a  a   a a   a
```

```
S ==> F ==> F ==>   f
  IO  |  IO | IO   / \
      G     a     a   a
      |
      a
```

On the other hand, the set of all possible OI

derivations is as follows:

```
S ==> F ==> f  ==>  f  ==>   f
  OI  | OI / \ OI  / \ OI   / \
      G   G   G   g   G   g     g
      |   |   |  / \  |  / \   / \
      a   a   a a   a a a   a a   a
```

```
S ==> F ==> f  ==>  f  ==>   f
  OI  | OI / \ OI  / \ OI   / \
      G   G   G   g   G   g     a
      |   |   |  / \  |  / \
      a   a   a a   a a a   a
```

```
S ==> F ==> f  ==>  f  ==>   f
  OI  | OI / \ OI  / \ OI   / \
      G   G   G   a   G   a     g
      |   |   |       |        / \
      a   a   a       a       a   a
```

```
S ==> F ==> f  ==>  f  ==>   f
  OI  | OI / \ OI  / \ OI   / \
      G   G   G   a   G   a     a
      |   |   |       |
      a   a   a       a
```

```
S ==> F ==> f  ==>  f  ==>     f
  OI  | OI / \ OI  / \ OI     / \
      G   G   G   G   g     g     g
      |   |   |   |  / \   / \   / \
      a   a   a   a a   a a   a a   a
```

```
S ==> F ==> f  ==>  f  ==>   f
  OI  | OI / \ OI  / \ OI   / \
      G   G   G   G   g    a   g
      |   |   |   |  / \      / \
      a   a   a   a a   a    a   a

S ==> F ==> f  ==>  f  ==>   f
  OI  | OI / \ OI  / \ OI   / \
      G   G   G   G   a    g   a
      |   |   |   |       / \
      a   a   a   a      a   a

S ==> F ==> f  ==>  f  ==>   f
  OI  | OI / \ OI  / \ OI   / \
      G   G   G   G   a    a   a
      |   |   |   |
      a   a   a   a
```

Also, the set of all possible (unrestr

is as follows:

```
S ==> F ==> f  ==>  f  ==>   f
      |    / \     / \      / \
      G   G   G   g   G    g   g
      |   |   |  / \  |   / \ / \
      a   a   a a   a a  a  a a  a

S ==> F ==> f  ==>  f  ==>   f
      |    / \     / \      / \
      G   G   G   g   G    g   a
      |   |   |  / \  |   / \
      a   a   a a   a a  a   a

S ==> F ==> f  ==>  f  ==>   f
      |    / \     / \      / \
      G   G   G   a   G    a   g
      |   |   |       |       / \
      a   a   a       a      a   a

S ==> F ==> f  ==>  f  ==>   f
      |    / \     / \      / \
      G   G   G   a   G    a   a
      |   |   |       |
      a   a   a       a
```

```
S ==> F ==> f    ==>  f    ==>    f
      |    / \       / \         / \
      G   G   G     G   g       g   g
      |   |   |     |  / \     / \ / \
      a   a   a     a  a a     a a a a

S ==> F ==> f    ==>  f    ==>    f
      |    / \       / \         / \
      G   G   G     G   g       a   g
      |   |   |     |  / \         / \
      a   a   a     a a   a       a   a

S ==> F ==> f    ==>  f    ==>    f
      |    / \       / \         / \
      G   G   G     G   a       g   a
      |   |   |     |          / \
      a   a   a     a         a   a

S ==> F ==> f    ==>  f    ==>    f
      |    / \       / \         / \
      G   G   G     G   a       a   a
      |   |   |     |
      a   a   a     a

S ==> F ==> F    ==>    f
      |     |          / \
      G     g         g   g
      |    / \       / \ / \
      a   a   a     a  a a  a

S ==> F ==> F    ==>  f
      |     |        / \
      G     a       a   a
      |
      a
```

Note that under an IO derivation, it is impossi

generate the trees

```
     f                f
    / \              / \
   a   g            g   a
      / \          / \
     a   a        a   a
```

ch can be generated by an OI derivation.

rthermore, at least for this example, the set of ees generated under an OI derivation is the same der the unrestricted case. It turns out that the sults are true in general, and are stated explici theorem 4.1.1. Before stating these results wever, the definition of a tree language must be tended to allow the derivation mode to be specifi

For notational convenience, the transitive osures of the different derivation modes are defi follows. The transitive closure of $\Longrightarrow$, $\underset{IO}{\Longrightarrow}$, an $\Rightarrow$ are denoted as $\Longrightarrow^+$, $\underset{IO}{\Longrightarrow}^+$, and $\underset{OI}{\Longrightarrow}^+$ respectivel ile the transitive reflexive closures of $\Longrightarrow$, $\underset{IO}{\Longrightarrow}$ d $\underset{OI}{\Longrightarrow}$ are denoted as $\Longrightarrow^*$, $\underset{IO}{\Longrightarrow}^*$, and $\underset{OI}{\Longrightarrow}^*$ spectively.

To extend the notion of a tree language under ther OI, IO, or unrestricted derivations, one mus so generalize the definition of sentential forms. ven a tree grammar $G=(\bar{\Phi},\bar{\Sigma},P,F_1)$, a derivation lation $\underset{R}{\Longrightarrow}^*$ where $R\in\{IO,OI,u\}$, and a sequence of ees $b_1,\ldots,b_{a_1}\in T_{\bar{\Sigma}}$, a <u>sentential form</u> is any tree $T_{\bar{\Sigma}\vee\bar{\Phi}}$ such that $F_1(b_1,\ldots,b_{a_1}) \underset{R}{\Longrightarrow}^* t$. Furthermo e <u>tree language generated by G using</u> $\underset{R}{\Longrightarrow}^*$, denote $(G,F_1(b_1,\ldots,b_{a_1}))$, is the set $\mid F_1(b_1,\ldots,b_{a_1}) \underset{R}{\Longrightarrow}^* t$ and $t\in T_{\bar{\Sigma}}\}$. Having

neralized these definitions, the following resul
ıglefriedt and Schmidt[77,78] is presented withou
oof:

heorem 4.1.1: Given a tree grammar $G=(\Phi,\Sigma,P,F_1)$ a
equence of trees $b_1,\ldots,b_{a_1} \in T_{\Sigma}$, the tree language
enerated from the three different derivation rela
re related as follows:

$$L_{IO}(G,F_1(b_1,\ldots,b_{a_1}))$$
$$\subseteq L_{OI}(G,F_1(b_1,\ldots,b_{a_1}))$$
$$= L_u(G,F_1(b_1,\ldots,b_{a_1})).$$

The remainder of this thesis will mainly deal
I derivations, since the class of tree languages
enerated by unrestricted derivations and the clas
enerated by OI derivations are identical. The re
o focus on OI derivations, as mentioned earlier,
hat they introduce restrictions which reduce the
umber of cases that need be considered in proofs.
nfortunately, the restrictions only correspond to
artial ordering, and hence the remainder of this
ection introduces an unconventional form of a OI
ne-step derivation such that a total ordering can
ssumed. That is, the notion of an OI derivation
lower bound u, as well as an OI derivation under
refix lexicographic ordering on tree domains are

introduced. Furthermore, it will be shown tha
tree language generated by an OI derivation un
prefix lexicographic ordering for tree address
equal to that generated by an OI derivation.

An <u>OI one-step derivation with lower boun</u>
(denoted $\overset{u}{\Longrightarrow}_G$) is an OI one-step derivation wh
be applied at any tree address v where $u \leq v$. G
tree grammar $G=(\Phi,\Sigma,P,F_1)$, the $\overset{u}{\Longrightarrow}_G$ relation i
as follows:

For any two trees $t_1, t_2 \in T_{\Sigma \vee \Phi}(X_A)$, $t_1 \overset{u}{\Longrightarrow}_G$ t
only if $t_1 \underset{OI}{\Longrightarrow} t_2$ and

i) $t_1 = s[v \leftarrow F(s_1,\ldots,s_n)]$,

ii) $t_2 = s[v \leftarrow t(s_1,\ldots,s_n)]$,

iii) $F(\vec{x}) \rightarrow t$ is an production in P where r(F

iv) $u \leq v$

Similarly, an <u>OI one-step derivation unde</u>
<u>prefix lexicographic ordering on tree domains</u>
$\overset{1}{\Longrightarrow}_G$) is an OI derivation applied to the
lexicographically smallest tree address labele
nonterminal. The $\overset{1}{\Longrightarrow}_G$ relation is defined as

For any two trees $t_1, t_2 \in T_{\Sigma \vee \Phi}(X_A)$, $t_1 \overset{1}{\Longrightarrow}_G$ t
only if $t_1 \underset{OI}{\Longrightarrow} t_2$ and

.) $t_1 = s[u \leftarrow F(s_1,\ldots,s_n)]$

.) $t_2 = s[u \leftarrow t(s_1,\ldots,s_n)]$

.) $F(\vec{x}) \to t$ is a production in P where $r(F)=n$, an

r) for all $v<u$, $t_1(v) \notin \Phi$.

For convenience, whenever the context of G is
vn, $\overset{u}{\Longrightarrow}_G$ and $\overset{1}{\Longrightarrow}_G$ will be denoted as $\overset{u}{\Longrightarrow}$ and $\overset{1}{\Longrightarrow}$
pectively.

In order to show that one can commute derivatio
least to some extent) when they are applied to
ependent subtrees, the following lemma is present

ma 4.1.1: Given a tree grammar $G_1 = (\Phi,\Sigma,P,F_1)$, a
ee trees $t_1,t_2,t_3 \in T_{\Sigma\vee\Phi}(X_A)$, if $t_1 \overset{u}{\Longrightarrow}^n t_2 \underset{OI}{\Longrightarrow} t_3$
re $t_2 = s[v \leftarrow F(s_1,\ldots,s_q)]$,
$= s[v \leftarrow t(s_1,\ldots,s_q)]$, $r(F)=q$, and $v<u$, then the
sts a tree $t_2' \in T_{\Sigma\vee\Phi}(X_A)$ such that $t_1 \underset{OI}{\Longrightarrow} t_2' \overset{u}{\Longrightarrow}^n$ t
re $t_1 = s'[v \leftarrow F(s_1,\ldots,s_q)]$ and
$= s'[v \leftarrow t(s_1,\ldots,s_q)]$.

roof: By induction on n.

ase case: $t_1 \xRightarrow[OI]{} t_3$. Trivial.

nductive step: $t_1 \xRightarrow{u} t_2 \xRightarrow{u}^{n} t_3 \xRightarrow[OI]{} t_4$ such tha

$_1 = s[w \leftarrow F(s_1,\ldots,s_q)]$, $t_2 = s[w \leftarrow t(s_1,\ldots,s_q$

$_3 = s'[v \leftarrow G(s'_1,\ldots,s'_{q'})]$,

$_4 = s'[v \leftarrow t'(s'_1,\ldots,s'_{q'})]$, $r(F)=q$, $r(G)=q'$, ar

$<u\underline{<}w$. By induction, $t_2 \xRightarrow[OI]{} t'_3 \xRightarrow{u}^{n} t_4$ such that

$_2 = s''[v \leftarrow G(s'_1,\ldots,s'_{q'})]$ and

$'_3 = s''[v \leftarrow t'(s'_1,\ldots,s'_{q'})]$. By the definition c

$\xRightarrow[I]{}$, v cannot be a prefix of w. But then, from th

efinition of tree substitution, there exists a tr

uch that

$_1 = t''[v \leftarrow G(s'_1,\ldots,s'_{q'})][w \leftarrow F(s_1,\ldots;s_q)]$,

$_2 = t''[v \leftarrow G(s'_1,\ldots,s'_{q'})][w \leftarrow t(s_1,\ldots,s_q)]$, ar

$'_3 = t''[v \leftarrow t'(s'_1,\ldots,s'_{q'})][w \leftarrow t(s_1,\ldots,s_q)]$.

he definition of $\xRightarrow[OI]{}$,

$_1 = t''[v \leftarrow G(s'_1,\ldots,s'_{q'})][w \leftarrow F(s_1,\ldots,s_q)] \xRightarrow[OI]{}$

$''[v \leftarrow t'(s'_1,\ldots,s'_{q'})][w \leftarrow F(s_1,\ldots,a_q)] = t'_2$.

he definition of $\xRightarrow{u}$,

$'_2 = t''[v \leftarrow t'(s'_1,\ldots,s'_{q'})][w \leftarrow F(s_1,\ldots,s_q)] \Rightarrow$

$''[v \leftarrow t'(s'_1,\ldots,s'_{q'})][w \leftarrow t(s_1,\ldots,s_q)] = t'_3$.

herefore $t_1 \xRightarrow[OI]{} t'_2 \xRightarrow{u}^{n+1} t_4$.

Using this result, one can show that every 01 jrivation (to a terminal tree) can be converted t< L derivation under a prefix lexicographic orderinj ; shown by the following lemma:

tmma 4,1,2: Given a tree grammar G-CJjij PjF.), a JO trees t_1 and t_2 such that t_1 G^yy JJ(XA) an<* 7^^ = $t_1 \overrightarrow{OI}^n t_2$, then fc; $\xrightarrow{1}{}_n$ t_2.

roof; By induction on n.

ise cases: n*0 and n«l« Both are trivial.

iductive step: $t_1 \overrightarrow{OI} t_3 \overrightarrow{OI}^n t_2$ where nM, L *s[u <- F(s1>..#,sq)]f t^ - s[u <- t(slf..«fsq id F(lt)->t6P. Depending on whether or not there cists a node w labeled by a nonterminal such that \ere are two cases:

ise 1: there does not exists a w in domCt1) such Cu and t1.(w)€l. But then $t_1 \xrightarrow{1} t_3$. By inductio } $\xrightarrow{1}{}^n$ t_2. Therefore tj =^>n+1 tj.

ise ^: there exists a w in dom(t1) such that w<u ,(w)€j. Let v€dom(tj) be the least tree address lat for all y'€dom(t1), where t1(y')€i, vj<y • By »finition of ^?>> clearly v is not a prefix of u. len it must be the case that t1. =^>m t4 OI=> t5 OI= lere m+p=n, m^O, t4 * s'[v <- G(s'1,..,8'q)], t5 - s'[v <- t'(s^f...f8^)], q'«r(G), G(X)->t€P,

is the least tree address such that $v<y$ and for a
$y'\in dom(t_1)$ such that $t_1(y')\in\Phi$, $y\leq y'$. By lemma 4.
$t_1 \underset{OI}{\Longrightarrow} t_6 \overset{y}{\Longrightarrow}^m t_5$ where $t_1 = s"[v \leftarrow G(s'_1,\ldots,s'_q$
and $t_6 = s"[v \leftarrow t'(s'_1,\ldots,s'_{q'})]$. But then $t_1 \overset{1}{\Longrightarrow}$
By induction, $t_6 \overset{1}{\Longrightarrow}^n t_2$. Hence $t_1 \overset{1}{\Longrightarrow}^{n+1} t_2$.

Using this result, it is easy to show that t
tree language generated by an OI derivation is eq
the language generated by an OI derivation under
prefix lexicographic ordering. In other words, g
any tree grammar $G=(\Phi,\Sigma,P,F_1)$ the <u>tree language u</u>
<u>an OI derivation under prefix lexicographic order</u>
<u>tree domains</u>, denoted $L^1_{OI}(G,F_1(b_1,\ldots,b_{a_1}))$ where
$b_1,\ldots,b_{a_1}\in T_\Sigma$, is the set
$\{t \mid F_1(b_1,\ldots,b_{a_1}) \overset{1}{\Longrightarrow}^* t$ where $t\in T_\Sigma\}$.

<u>**Theorem 4.1.2:**</u> Given a tree grammar $G=(\Phi,\Sigma,P,F_1)$
any set of trees $b_1,\ldots,b_{a_1}\in T_\Sigma$,
$L^1_{OI}(G,F_1(b_1,\ldots,b_{a_1})) = L_{OI}(G,F_1(b_1,\ldots,b_{a_1}))$.

<u>**Proof:**</u> Inspecting the definition of $\overset{1}{\Longrightarrow}$, it is cl
that if $S \overset{1}{\Longrightarrow}^* t$, then $S \underset{OI}{\Longrightarrow}^* t$. Hence,
$L^1_{OI}(G,F_1(b_1,\ldots,b_{a_1})) \subseteq L_{OI}(G,F_1(b_1,\ldots,b_{a_1}))$. O
other hand, if $S \underset{OI}{\Longrightarrow}^* t$, where $t\in T_\Sigma$, then by lemma
$S \overset{1}{\Longrightarrow}^* t$. Thus

$(G,F_1(b_1,...,b_{a_1})) \subseteq L^1_{OI}(G,F_1(b_1,...,b_{a_1}))$, and

ce $L_{OI}(G,F_1(b_1,...,b_{a_1})) = L^1_{OI}(G,F_1(b_1,...,b_{a_1}))$.

Using the above theorem, and theorem 4.1.1, it ar that the tree language generated using an estricted derivation is identical to the tree guage generated using an OI derivation under a fix lexicographic ordering. For the remainder of s thesis, all proofs will use OI derivations unde fix lexicographic ordering on tree domains. Henc never an OI derivation is used in a proof (i.e. ation $\xRightarrow[OI]{}$) it will be implicitly assumed that in t it is an OI derivation under a prefix icographic ordering.

Example 4.1.4: Using the tree grammar G_3 in examp 4.1.3, the set of all OI derivations under a pref lexicographic ordering is as follows:

```
S =1=> F =1=>  f   =1=>  f   =1=>    f
       |      / \       / \         / \
       G     G   G     g   G       g   g
       |     |   |    / \  |      / \ / \
       a     a   a a   a a  a     a a  a

S =1=> F =1=>  f   =1=>  f   =1=>    f
       |      / \       / \         / \
       G     G   G     g   G       g   a
       |     |   |    / \  |      / \
       a     a   a a   a a  a     a
```

```
S =1=> F =1=>  f   =1=>  f   =1=>   f
       |      / \       / \        / \
       G     G   G     a   G      a   g
       |     |   |         |         / \
       a     a   a         a        a   a

S =1=> F =1=>  f   =1=>  f   =1=>   f
       |      / \       / \        / \
       G     G   G     a   G      a   a
       |     |   |         |
       a     a   a         a
```

Note that the set of trees generated
as the set generated under an OI derivati
Furthermore, the total number of possible
is reduced, indicating the point alluded
That is, the number of cases needed to co
proof should be reduced, since there are
derivations for any given tree language.

4.2 Augmented Tree Grammars

One of the problems with tree langua
general, is that they are parameterized (
supply a sequence of trees $b_1,\ldots,b_{a_1}$ alo
tree grammar). It would be more convenie
arity of the start symbol could be reduce
hence no parameters would be necessary.
that this is possible by augmenting the t
with a new start symbol with arity zero.
the augmentation will maintain the tree l

ıerated. Given a tree grammar $G_1=(\Phi_1,\Sigma,P_1,F_1)$, a
ıuence of trees $b_1,\ldots,b_{a_1} \in T_{\Sigma}$, an augmented tree
ımmar G_2, denoted aug$(G_1,(b_1,\ldots,b_{a_1}))$, is the tu
$=(\Phi_2,\Sigma,P_2,S)$ where

$\Phi_2 = \Phi_1 \cup \{S\}$ where $S \notin \Phi_1 \cup \Sigma$; and

$P_2 = P_1 \cup \{S \rightarrow F_1(b_1,\ldots,b_{a_1})\}$.

other words, the augmented tree grammar G_2 is th
ee grammar G_1 with an added auxiliary start
oduction. Furthermore, the auxiliary start
oduction is not based on any "outside" parameters
.e. the tree grammar is totally defined).

A natural assumption is to believe that the ab
ansformation does not alter the tree language
nerated. This is shown to be true by the followi
eorem:

eorem 4.2.1: Given a tree grammar $G_1=(\Phi_1,\Sigma,P_1,F_1)$
t of trees $b_1,\ldots,b_{a_1} \in T_{\Sigma}$, and
$=$aug$(G_1,(b_1,\ldots,b_{a_1}))$, then
$_I(G_1,F_1(b_1,\ldots,b_{a_1})) = L_{OI}(G_2,S)$.

oof: Left as an exercise for the reader.

Since every tree grammar can be augmented wit ffecting the tree language generated, the remaind his thesis will assume that all tree grammars are ugmented. Furthermore, since the start symbol ha rity zero, $L_{OI}(G,S)$, $L_{IO}(G,S)$, and $L_u(G,S)$ will b enoted as $L_{OI}(G)$, $L_{IO}(G)$, and $L(G)$ respectively w $=(\Phi,\Sigma,P,S)$.

.3 Redundant Tree Grammars

A production p is <u>redundant</u> if p is of the fo $(\vec{x})\rightarrow F(\vec{x})$. Similarly, a tree grammar $G_1=(\Phi,\Sigma,P_1,$ <u>edundant</u> if there exists a production $p\in P_1$ such t s redundant. In other words, whenever $t_1 \overset{}{\underset{OI}{\Longrightarrow}} t_2$ $(\vec{x})\rightarrow F(\vec{x})$, $t_1=t_2$. Hence, there is no need for th roduction $F(\vec{x})\rightarrow F(\vec{x})$. The <u>nonredundant</u> <u>tree</u> <u>gram</u> <u>f</u> G_1, denoted $nr(G_1)$, is a tree grammar $G_2=(\Phi,\Sigma,P$ here $P_2 = P_1 - \{F(\vec{x})\rightarrow F(\vec{x})\in P_1\}$.

Like augmented tree grammars, redundant tree rammars can be converted to nonredundant tree gra nd the transformation does not alter the tree lan enerated. Without proof, this fact is stated by ollowing theorem:

Theorem **4.3.1:** Given a tree grammar $G_1=(\Phi,\Sigma,P_1,$
$G_2=nr(G_1)$, $L_{OI}(G_1) = L_{OI}(G_2)$.

The remainder of this thesis will assume t
tree grammars are not redundant. Furthermore,
transformation presented in the remainder of th
thesis introduces a redundant tree grammar, the
implicitly assumed that the actual tree gramma
generated is the nonredundant version of the tr
grammar generated.

4.4 NT/T Segmented Grammars

NT/T segmented grammars are tree grammars
that each production's right-hand side is eithe
labeled using nonterminal symbols, or terminal
but not both. Furthermore, if the right-hand s
production is labeled by terminal symbols, the
is only one terminal symbol in the tree. Howev
order to present NT/T segmented grammars forma
new terminology which allows transforming term
symbols to nonterminal symbols must be introdu

let $\bar{\Sigma}$ and Γ be two ranked alphabets such that

i) $\bar{\Sigma} \wedge \Gamma = \emptyset$

ii) $|\bar{\Sigma}| = |\Gamma|$

iii) There exists a bijective function pi : $\bar{\Sigma}$ -

iv) For all $a \in \bar{\Sigma}$ $r(a) = r(pi(a))$

en, pi is a <u>renaming function</u> of $\bar{\Sigma}$ using Γ, deno $\Gamma = pi(\bar{\Sigma})$. In general, renaming will be used to eate a new unique nonterminal symbol for each rminal symbol in the tree grammar.

Extending the renaming function pi, if $\bar{\Sigma}$, $\bar{\Phi}$, a e three ranked alphabets such that $\Gamma = pi(\bar{\Sigma})$ and $\wedge \bar{\Phi} = \emptyset$, let $pi^* : T_{\bar{\Sigma} \vee \bar{\Phi} \vee \Gamma}(X_m) \rightarrow T_{\Gamma \vee \bar{\Phi}}(X_m)$ where $= \max\{r(f) \mid f \in \bar{\Sigma} \vee \Gamma\}$, be a function such that gi y tree $t \in T_{\bar{\Sigma} \vee \bar{\Phi} \vee \Gamma}(X_m)$:

i) $dom(pi^*(t)) = dom(t)$

ii) For all $u \in dom(pi^*(t))$, if $t(u) \in \bar{\Sigma}$, then
$pi^*(t)(u) = pi(t(u))$, otherwise
$pi^*(t)(u) = t(u)$.

t $pi^n : T_{\bar{\Sigma} \vee \bar{\Phi}}(X_m) \rightarrow 2^{T_{\bar{\Sigma} \vee \bar{\Phi} \vee \Gamma}(X_m)}$ be a function cursively defined such that given any tree $T_{\bar{\Sigma} \vee \bar{\Phi}}(X_m)$,

i) $pi^0(t) = \{t\}$

ii) $pi^{i+1}(t) = \{s \in pi^i(t)$

$\mid \exists u \in dom(t)$ such that $t(u) \in \bar{\Sigma}\}$

$\vee \{s[u \leftarrow pi(s(u))(s/u1,\ldots,s/uq)]$

$\mid s \in pi^i(t),\ u \in dom(s),\ s(u) \in \bar{\Sigma},$

and $r(s(u))=q\}$ for any $i \geq 0$.

thermore, let $pi^{-1} : T_{\bar{\Sigma} \vee \Phi \vee \Gamma}(X_m) \rightarrow T_{\bar{\Sigma} \vee \Phi}(X_m)$ be
ction such that given any tree $t \in T_{\bar{\Sigma} \vee \Phi \vee \Gamma}(X_m)$,

i) $dom(pi^{-1}(t)) = dom(t)$

ii) For all $u \in dom(t)$, if $t(u) \in \Gamma$ such that $pi(f)=t(u)$ for some $f \in \bar{\Sigma}$, then $pi^{-1}(t)(u)=f$, otherwise $pi^{-1}(t)(u)=t(u)$.

other words, $pi^*(t)$ is the tree t where all nodes
eled by $\bar{\Sigma}$ are renamed by their corresponding symb
Γ. If $1_{\bar{\Sigma}}(t) \leq n$, then $pi^n(t)$ is the set $\{pi^*(t)\}$.
erwise, $pi^n(t)$ is the set of trees obtained from
sible conversions of n nodes labeled by terminal
bols in $\bar{\Sigma}$, to nonterminal symbols in Γ. Finally
en any tree $t \in pi^n(t')$ where $t' \in T_{\bar{\Sigma} \vee \Phi}(X_A)$,
$^{1}(t)=t'$. In other words, to some extent pi^{-1} is
erse function of pi^n and pi^*. These results are
wn by the following lemmas.

emma 4*4,1: Given three ranked alphabets i, (jj, ar uch that r= pi(2), P A i = 0, any n>0 , and any tr $_1$€T∑VΦ(X$_m$) where m = max{r(f) | f€iVffi}, then for ree t ^pi^n(t$_1$)

i) if n a ^ t p , then lp(t$_2$) = n,

$l_\Sigma(t_2)$ = $l_\Sigma(t_1)$-n, and pi*(t$_2$)-pi*(t$_1$)

ii) if n>l^(t$_1$), then lp(t$_2$) = $l_\Sigma(t_1)$, $l_\Sigma(t_2$

and t^^pi*(t$_1$)

roof s by induction on n.

ase case; n * 0* Trivial. By the definition of i^0(t$_i$) - {t$_i$.}. Hence, it must be the case that i $_2$€pi°(t$_1$), t$_2$-tj. By the definition of pi(I), ΛΣ - 0, and we were given that f Af * 0« Hence ince ^{t}i$^{€T}$2\/i^^{X}m^^{J} ^{1}p ^ ^{t} 2 ^ = °* Furthermore, it ti e the case that n <^ -•s"^^{t}l^ since n = 0. Finally, ^(t$_2$) - l^Ctj) and pi*(t$_2$) - pi*(tj) since tj - t

nductive step: Assume the hypothesis is true foi <JI for some n. We want to show that the lemma is or any t^Spi^{11+1}(t$_i$). According to the definition i^{n+1}, there are two possibilities:

ase 1: t^€pi^n(t$_i$) and there does not exist a €dom(t$_2$) such that t$_2$(u)€I. Clearly, l>;(t$_2$) - 0.

$_{\Sigma}(t_1)$, then by induction on condition i), $1_{\Sigma}(t_2)$
ch contradicts that $1_{\Sigma}(t_2)=0$. Hence, $n \geq 1_{\Sigma}(t_1)$.
$_{\Sigma}(t_1)$, by induction using condition i), $1_{\Sigma}(t_2)=0$
$(t_2)=1_{\Sigma}(t_1)$, and $t_2=pi^*(t_2)=pi^*(t_1)$ in which case
isfies condition ii) for the n+1 case. Finally,
$_{\Sigma}(t_1)$, by induction on condition ii), $1_{\Gamma}(t_2)=1_{\Sigma}($
$t_2)=0$, and $t_2=pi^*(t_2)=pi^*(t_1)$. Hence, for any t
$pi^{n+1}(t_1)$, when $t_2 \in pi^n(t_1)$, condition ii) is
isfied for the n+1 case.

e 2: $t_2=s[u$ <- $pi(s(u))(s/u1,...,s/uq)]$ where
$i^n(t_1)$, $u \in dom(s)$, $s(u) \in \overline{\Sigma}$, and $r(s(u))=q$. Since
$) \in \overline{\Sigma}$, $1_{\Sigma}(s)>0$. Using induction, condition i) mus
ly, and hence, $n \leq 1_{\Sigma}(t_1)$, $1_{\Gamma}(s)=n$, $1_{\Sigma}(s)=1_{\Sigma}(t_1)-$
l $pi^*(s)=pi^*(t_1)$. Also, by the tree substitution
formed at tree address u, to construct t_2, adds
e node labeled by Γ previously labeled by Σ. He
$(t_2)=1_{\Sigma}(s)-1$, $1_{\Gamma}(t_2)=1_{\Gamma}(s)+1$, and $pi^*(t_2)=pi^*(s)$
then, $1_{\Gamma}(t_2)=n+1$, $1_{\Sigma}(t_2)=L_S(s)-(n+1)$, and
$^*(t_2)=pi^*(t_1)$ All that is left to show is that
$1 \leq 1_{\Sigma}(t_1)$. But since $1_{\Sigma}(t_1)-n>0$, $1_{\Sigma}(t_1)-(n+1) \geq 0$,
$(t_1) \geq n+1$.

mma **4.4.2:** Given three ranked alphabets $\overline{\Sigma}$, $\overline{\Phi}$, and ch that $\Gamma=pi(\overline{\Sigma})$ and $\Gamma \wedge \overline{\Phi}=\emptyset$, and a tree $t \in T_{\overline{\Sigma} \vee \overline{\Phi}}(X_m$ ere $m=\max\{r(f) \mid f \in \overline{\Sigma} \vee \overline{\Phi}\}$, then for any $n \geq 1_{\overline{\Sigma}}(t)$, $^{*}(t) \in pi^{n}(t)$ and $|pi^{n}(t)|=1$.

oof: Using lemma 4.4.1, for any tree $t_1 \in pi^{n}(t)$, $(t_1)=0$ and $pi^{*}(t_1)=pi^{*}(t)$. Since $1_{\overline{\Sigma}}(t_1)=0$, it mu the case that $t_1 \in T_{\Gamma \vee \overline{\Phi}}(X_m)$. By inspection of th finition of pi^{*}, clearly $pi^{*}(t_1)=t_1$. But then $^{*}(t) \in pi^{n}(t)$, and $|pi^{n}(t)| \geq 1$. To show that $i^{n}(t)|=1$, assume there exists a tree $t_2 \in pi^{n}(t)$ su at $t_2 \neq pi^{*}(t)$. Using lemma 4.4.1, $1_{\overline{\Sigma}}(t_2)=0$ and $^{*}(t_2)=pi^{*}(t)$. Since $t_2 \neq pi^{*}(t)$, and $pi^{*}(t_2)=pi^{*}(t$ ere must exist a node $u \in dom(t_2)$ such that $t_2(u) \in \overline{\Sigma}$ t this is impossible since $1_{\overline{\Sigma}}(t_2)=0$. Hence is mu the case that $|pi^{n}(t)|=1$.

mma **4.4.3:** Given three ranked alphabets $\overline{\Sigma}$, $\overline{\Phi}$, and ch that $\Gamma=pi(\overline{\Sigma})$ and $\Gamma \wedge \overline{\Phi}=\emptyset$, and any tree $t_1 \in T_{\overline{\Sigma} \vee \overline{\Phi}}$ ere $m=\max\{r(f) \mid f \in \overline{\Sigma} \vee \overline{\Phi}\}$, then for any $n \geq 0$, if $\in pi^{n}(t_1)$, then $pi^{-1}(t_2)=t_1$.

roof: By induction on n.

ase case: $t_2 \in pi^0(t_1)$. By the definition of pi^0, $i^0(t_1)=\{t_1\}$. Hence, $t_2=t_1$. Also, since $t_1 \in T_{\overline{\Sigma}V\overline{\Phi}}($ $i^{-1}(t_1)=t_1$. Hence, $pi^{-1}(t_2)=t_1$.

nductive step: $t_2 \in pi^{n+1}(t_1)$. By the definition i^{n+1}, there are two possibilities:

ase 1: $t_2 \in pi^n(t_1)$ and for all $u \in dom(t_2)$, $t(u) \notin \overline{\Sigma}$. rivial. By induction, $pi^{-1}(t_2)=t_1$.

ase 2: $t_2=s[u \leftarrow pi(s(u))(s/u1,\ldots,s/uq)]$ where $\in pi^n(t_1)$, $u \in dom(s)$, $s(u) \in \overline{\Sigma}$, and $r(s(u))=q$. By nduction, $pi^{-1}(s)=t_1$. By inspection of the defin f pi^{-1}, clearly

$i^{-1}(t_2)=t_1[u \leftarrow s(u)(t_1/u1,\ldots,t_1/uq)]=t_1$.

Having completed the above terminology, NT/T egmented grammars can be formally introduced. Gi ree grammar $G = (\overline{\Phi},\overline{\Sigma},P,S)$, a production $F(\vec{x}) \rightarrow t \in P$ T/T segmented if and only if either

i) For all $u \in (dom(t)-var(t))$, $t(u) \in \overline{\Phi}$

ii) $t(\varepsilon) \in \overline{\Sigma}$ and $(dom(t)-var(t))=\{\varepsilon\}$

n other words, condition (i) states that every no ot labeled by a variable is labeled with a onterminal, and condition (ii) states that the ro beled with a terminal symbol and each of its mediate descendants are labeled by variables. milarly, a tree grammar $G=(\bar{\Phi},\Sigma,P,S)$ is <u>NT/T segme</u> and only if for every $p\in P$, p is NT/T segmented.

Example 4.4.1: Let $G_1=(\bar{\Phi}_1,\Sigma,P_1,S)$ and $G_2=(\bar{\Phi}_2,\Sigma,P$ such that

$\bar{\Phi}_1 = \{S,F\}$ where $r(S)=0$ and $r(F)=1$;

$\Sigma = \{a,f\}$ where $r(a)=0$ and $r(f)=1$;

```
P  = {S->F , F -> f },
 1      |    |   / \
        a    x  x   x
```

$\bar{\Phi}_2 = \{S,F,\hat{a},\hat{f}\}$ where $r(S)=r(\hat{a})=0$, $r(F)=1$, anc $r(\hat{f})=2$; and

```
P  = {S -> F , F -> f̂ , â -> a ,
 2          |    |    / \
            â    x   x   x

     f̂ -> f }.
    / \  / \
   x   y x  y
```

Note that while $L_{OI}(G_1) = L_{OI}(G_2)$, G_1 is not NT/ segmented (because of the production

```
S -> F ),
     |
     a
```

while G_2 is NT/T segmented.

It is natural question to ask if there is an orithm to convert a tree grammar G_1 into a new tr mmar G_2 such that G_2 is NT/T segmented and $(G_1)=L_{OI}(G_2)$. The answer is yes as is shown by t lowing definition and lemmas.

Given a context-free tree grammar $G_1=(\Phi_1,\Sigma,P_1,S$ $\Gamma=pi(\Sigma)$ such that $\Gamma\wedge\Phi=\emptyset$, let $G_2=(\Phi_2,\Sigma,P_2,S)$ be T <u>segmented grammar of G_1</u>, denoted $NT/T(G_1)$, wher

i) $\Phi_2 = \Phi_1 \vee \Gamma$

ii) $P_2 = \{pi(f)(\vec{x})\text{->}f(\vec{x}) \mid f\in\Sigma\}$
$\qquad\qquad \vee\{F(\vec{x})\text{->}pi^*(t) \mid F(\vec{x})\text{->}t\in P_1\}$

<u>Example 4.4.2:</u> Let G_1 and G_2 be defined as in example 4.4.1. Then, G_2 is the NT/T segmented grammar of G_1 where $\Gamma=\{\hat{a},\hat{f}\}$, $pi(a)=\hat{a}$, and $pi(f)=$

<u>ma 4.4.4:</u> Given any two tree grammars G_1 and G_2 s t $G_1=(\Phi_1,\Sigma,P_1,S)$ and $G_2=NT/T(G_1)$, G_2 is NT/T mented.

<u>of:</u> According to the definition of P_2, there are sible forms of productions:

<u>e</u> 1: $pi(f)(\vec{x})\text{->}f(\vec{x})$ where $f\in\Sigma$. For the tree $f(\vec{x}$ arly $f(\vec{x})(\varepsilon)=f\in\Sigma$ and $(dom(f(\vec{x}))-var(f(\vec{x})))=\{\varepsilon\}$. ce $pi(f)(\vec{x})\text{->}f(\vec{x})$ is NT/T segmented.

<u>case</u> 2: $F(\vec{x})$->$pi^*(t)$ such that $F(\vec{x})$->$t \in P_1$. By definition of pi^*, $pi^*(t) \in T_{\bar{\Phi}_1 \vee \Gamma}(X_m)$. By the definition of G_2, $\bar{\Phi}_2 = \bar{\Phi}_1 \vee \Gamma$, and hence, for all $u \in (dom(pi^*(t)) - var(pi^*(t)))$, $pi^*(t)(u) \in \bar{\Phi}_2$.

Therefore, every production $p \in P_2$ is NT/T segmen hence G_2 is NT/T segmented.

<u>Lemma 4.4.5:</u> Given any two tree grammars G_1 and where $G_1 = (\bar{\Phi}_1, \bar{\Sigma}, P_1, S)$ and $G_2 = NT/T(G_1)$, any tree $t_1 = s[u \leftarrow t(s_1, \ldots, s_q)] \in T_{\bar{\Sigma} \vee \bar{\Phi}_1 \vee \Gamma}(X_m)$ where $m = max\{r(f) \mid f \in \bar{\Sigma} \vee \bar{\Phi}_1\}$ and $t \in T_{\bar{\Sigma} \vee \bar{\Phi}_1}(X_q)$, then for $n \geq 0$, and any tree $t_2 = s[u \leftarrow t'(s_1, \ldots, s_q)]$ such $t' \in pi^n(t)$, $t_2 \overset{*}{\Longrightarrow}_{G_2} t_1$.

<u>Proof:</u> By induction on n.

<u>base case</u>: n=0 - Trivial.

<u>inductive step</u>: Let t′ be any tree in $pi^{n+1}(t)$ the definition of pi^{n+1}, two possibilities aris

<u>case</u> 1: $t' \in pi^n(t)$. By induction, clearly $t_2 \Longrightarrow$

<u>case</u> 2: $t' = s'[v \leftarrow pi(s'(v))(s'/v1, \ldots, s'/vq')$ $s' \in pi^n(t)$, $v \in dom(s')$, $s'(v) \in \bar{\Sigma}$, and $r(s'(v)) = q'$. definition of G_2, $pi(s'(v))(\vec{x})$->$s'(v)(\vec{x}) \in P_2$. C

$t_2 =$

$s[u \leftarrow s'[v \leftarrow pi(s'(v))(s'/v1, \ldots, s'/vq')](s_1,$

$\Longrightarrow_{G_2}$

$s[u \leftarrow s'[v \leftarrow s'(v)(s'/v1,\ldots,s'/vq')](s_1,\ldots,s_q$
$= s[u \leftarrow s'(s_1,\ldots,s_q)]$. Since $s' \in pi^n(t)$, by
induction, $s[u \leftarrow s'(s_1,\ldots,s_q)] \Longrightarrow^*_{G_2}$
$s[u \leftarrow t(s_1,\ldots,s_q)]=t_1$. Hence $t_2 \Longrightarrow^*_{G_2} t_1$.

Lemma 4.4.6: Given any two tree grammars G_1 and G
where $G_1=(\overline{\Phi}_1,\overline{\Sigma},P_1,S)$ and $G_2=NT/T(G_1)$, if $S \underset{\overline{OI}}{\Longrightarrow}{}^*_{G_1} t$
$S \Longrightarrow^*_{G_2} t$.

Proof: By induction on n.

base case: n=0 - Trivial.

inductive step: Assume $S \underset{\overline{OI}}{\Longrightarrow}{}^n_{G_1} t_1 \Longrightarrow_{G_1} t_2$ where
$t_1 = s[u \leftarrow F(s_1,\ldots,s_m)]$, $t_2 = s[u \leftarrow t(s_1,\ldots,s$
$m=r(F)$, and $F(\vec{x})\text{->}t \in P_1$. By induction, $S \Longrightarrow^*_{G_2} t_1$
the definition of G_2, $F(\vec{x})\text{->}pi^*(t) \in P_2$ and hence
$s[u \leftarrow F(s_1,\ldots,s_m)] \Longrightarrow_{G_2} s[u \leftarrow pi^*(t)(s_1,\ldots,s$
By lemma 4.4.2, $s[u \leftarrow pi^*(t)(s_1,\ldots,s_m)] =$
$s[u \leftarrow t_3(s_1,\ldots,s_m)]$ where $t_3=pi^*(t) \in pi^q(t)$ for
$q=l_{\Sigma}(t) \leq |dom(t)|$. By lemma 4.4.5,
$s[u \leftarrow t_3(s_1,\ldots,s_m)] \Longrightarrow^*_{G_2} s[u \leftarrow t(s_1,\ldots,s_m)]$.
Hence $S \Longrightarrow^*_{G_2} t_2$.

Lemma 4.4.7: Given any two tree grammars G_1 and G
where $G_1=(\overline{\Phi}_1,\overline{\Sigma},P_1,S)$ and $G_2=NT/T(G_1)$, any tree
$t \in T_{\Sigma \vee \overline{\Phi}_1 \vee \Gamma}(X_m)$ where $m=\max\{r(f) \mid f \in \overline{\Sigma} \vee \overline{\Phi}_1 \vee \Gamma\}$ and
$l_{\Gamma}(t)=n$, then there exists a unique tree $t' \in T_{\Sigma \vee \overline{\Phi}}$

uch that

i) $t \in pi^{n}(t')$

ii) $t' = pi^{-1}(t)$

iii) For all $u \in dom(t)$ if $t(u) \in \overline{\Sigma}$, then $t'(u)=t($

roof: By induction on n.

ase case: $l_{\Gamma}(t)=0$ - Trivial. $t'=t$.

nductive step: Assume the hypothesis is true for
$\leq n$. We want to show that for any t such that
$_{\Gamma}(t)=n+1$, the existence of a unique t' such that
$' = pi^{-1}(t)$. Since $l_{\Gamma}(t)>0$, there exists a $u \in do$
uch that $t(u) \in \Gamma$. Let $s \in \overline{\Sigma}$ be the symbol such tha
$i(s)=t(u)$. Furthermore, let t'' be the tree such

1) $dom(t'')=dom(t)$

2) for all $v \in dom(t'')$, if $v=u$, then $t''(v)=s$, otherwise $t''(v)=t(v)$.

learly, $l_{\Gamma}(t'')=n$. Hence, by induction, there ex
nique tree $t' \in T_{\overline{\Sigma} \vee \Phi_1}(\mathbf{X}_m)$ such that $t'' \in pi^{n}(t')$,
$'=pi^{-1}(t'')$, and for all $v \in dom(t)$, if $t'(v) \in \overline{\Sigma}$, the
$'(v)=t'(v)$. Hence $t'(u)=t''(u)=s(u)$. By the
efinition of pi^{n+1},
$=t''[u \leftarrow pi(s(u))(s/u1,\ldots,s/uq)]$, where $q=r(s(u)$

d hence $t \in pi^{n+1}(t')$. By lemma 4.4.3, clearly nditions 1) and 2) are met. For all $v \in dom(t'')$, i $(v) \in \bar{\Sigma}$, then $t''(v)=t'(v)$, and condition 3) is met.

mma 4.4.8: Given any two tree grammars G_1 and G_2 ere $G_1=(\bar{\Phi}_1,\bar{\Sigma},P_1,S)$ and $G_2=NT/T(G_1)$, any two trees $,t_2 \in T_{\bar{\Sigma}\vee\bar{\Phi}_1\vee\Gamma}(X_m)$ where $m=\max\{r(f) \mid f \in \bar{\Sigma}\vee\bar{\Phi}_1\vee\Gamma\}$, $\overline{\overline{OI}}\!>^{n}_{G_2}$ t_2, then there exists trees $t_3,t_4 \in T_{\bar{\Sigma}\vee\bar{\Phi}_1}(X$ $,n_2 \in \mathbf{N}$, such that $t_1 \in pi^{n_1}(t_3)$, $t_2 \in pi^{n_2}(t_4)$, and $\Longrightarrow^{*}_{G_1} t_4$.

oof: By induction on n.

se case: $n=0$. Hence, $t_1=t_2$. By lemma 4.4.7, th ists a unique tree $t'=pi^{-1}(pi^{q}(t))$ where $1_{\Gamma}(t_1)=$ t $t_3=t_4=t'$ and $n_1=n_2=q$. CLearly $t_3 \Longrightarrow^{*}_{G_1} t_4$, and nce the conditions are met.

ductive step: Assume $t_1 \overline{\overline{OI}}\!>_{G_2} t_1' \overline{\overline{OI}}\!>^{n}_{G_2} t_2$. Usin duction, let $t_3,t_4 \in T_{\bar{\Sigma}\vee\bar{\Phi}_1}(X_m)$ be the trees such t $\in pi^{n_1'}(t_3)$, $t_2 \in pi^{n_2}(t_4)$, and $t_3 \Longrightarrow^{*}_{G_1} t_4$. By the finition of an OI derivation, there must exist a $T_{\bar{\Sigma}\vee\bar{\Phi}_1\vee\Gamma}(X_m)$ such that

$=s[u \leftarrow F(\vec{x})(s/u1,\ldots,s/uq)]$ and

$=s[u \leftarrow t'(s/u1,\ldots,s/uq)]$ where $q=r(F)$, $F(\vec{x})\rightarrow t'$

d for all proper prefixes v of u, $s(v) \notin \bar{\Phi}_2$. Accor the definition of G_2, there are two possibilitie se 1:

$$\begin{tikzcd}
t_1 \arrow[r, Rightarrow, "\overline{OI}"' , "G_2"] \arrow[dr, "pi^{-1}"'] & t_1' \arrow[r, Rightarrow, "n", "\overline{OI}\ G_2"'] \arrow[d, "pi^{-1}"] & t_2 \arrow[d, "pi^{-1}"] \\
 & t_3 \arrow[r, Rightarrow, "*", "\overline{OI}\ G_1"'] & t_4
\end{tikzcd}$$

Γ and $F(\vec{x})\text{->}t'$ is of the form $pi(f)(\vec{x})\text{->}f(\vec{x})$ whe

Σ. Since $t_1'=s[u \leftarrow f(s/ul,\ldots,s/uq)]\in pi^{n_1'}(t_3)$ ar

spection of the definition of $pi^{n_1'+1}$, clearly we

$=s[u \leftarrow pi(f)(s/ul,\ldots,s/uq)]\in pi^{n_1'+1}(t_3)$. Hence

$\in pi^{n_1'+1}(t_3)$, $t_2\in pi^{n_2}(t_4)$ and $t_3 \Longrightarrow^*_{G_1} t_4$.

se 2:

$$\begin{tikzcd}
t_1 \arrow[r, Rightarrow, "\overline{OI}\ G_2"'] \arrow[d, "pi^{-1}"] & t_1' \arrow[r, Rightarrow, "n", "\overline{OI}\ G_2"'] \arrow[d, "pi^{-1}"] & t_2 \arrow[d, "pi^{-1}"] \\
t_3' \arrow[r, Rightarrow, "\overline{OI}\ G_1"'] & t_3 \arrow[r, Rightarrow, "*", "\overline{OI}\ G_1"'] & t_4
\end{tikzcd}$$

$=pi^*(t)$ where $F(\vec{x})\text{->}t\in P$. By lemma 4.4.7, there

ists a unique tree $s'=pi^{-1}(s)$ such that

$=s'[u \leftarrow t(s'/ul,\ldots,s'/uq)]$ where for all i, $1\leq i$

d $s'/ui=pi^{-1}(s/ui)$. Clearly, by the definition o

rivation, $t_3'=s'[u \leftarrow F(s'/u_1,\ldots,s'/uq)] \Longrightarrow_{G_1}$

$[u \leftarrow t(s'/ul,\ldots,s'/uq)]=t_3$. Hence

$\Longrightarrow^*_{G_1} t_3 \Longrightarrow^*_{G_1} t_4$. By inspection of the definit

pi^{-1}, clearly $t_3'=pi^{-1}(t_1)$. But then, by lemma

4.7, $t_1\in pi^{n_1}(t_3')$ for some $n_1\geq 0$, Hence the conditio the lemma are met.

eorem 4.4.1: Given any two tree grammars G_1 and G
ere $G_1=(\Phi_1,\Sigma,P_1,S)$ and $G_2=NT/T(G_1)$, $L_{OI}(G_1)=L_{OI}(G$

oof: By the definition of a tree language, $L_{OI}(G)$
e set of all trees $t \in T_{\Sigma}$ such that $S \overset{*}{\underset{OI}{\Longrightarrow}}_G t$. By
mma 4.4.6, we know that if $S \overset{*}{\underset{OI}{\Longrightarrow}}_{G_1} t$, then $S \overset{*}{\Longrightarrow}_G$
nce $L_{OI}(G_1) \subseteq L(G_2)$. By theorem 4.1.1,
$G_2)=L_{OI}(G_2)$. Hence $L_{OI}(G_1) \subseteq L_{OI}(G_2)$. On the ot
nd, if $S \overset{*}{\underset{OI}{\Longrightarrow}}_{G_2} t$, where $t \in T_{\Sigma}$, then by lemma 4.4.8
ere exists trees $t_1,t_2 \in T_{\Sigma \vee \bar{\Phi}_1}(X_m)$ such that
$pi^n1(t_1)$, $t \in pi^n2(t_2)$, and $t_1 \overset{*}{\Longrightarrow}_{G_1} t_2$. But $pi^{-1}($
d $pi^{-1}(t)=t$ since $t \in T_{\Sigma}$. Thus $S=t_1$ and $t=t_2$. Hen
$_I(G_2) \subseteq L(G_1)$. Using theorem 4.1.1, $L(G_1) = L_{OI}($
nce $L_{OI}(G_2) \subseteq L_{OI}(G_1)$. Therefore $L_{OI}(G_1) = L_{OI}(G$

5 **n - Normal Forms**

A tree grammar is in n - normal form if the nu
nodes labeled by terminal and nonterminal symbol
curring on the right-hand side of each productior
es not exceed n. Of interest here, is to show th
y tree grammar G_1 can effectively be transformed
equivalent context-free tree grammar G_2 (under a

lerivation) such that G_2 is in 2-normal form. How
Ln order to show this, some terminology has to be
>resented.

Given a ranked alphabet Σ, and $n \geq 0$, let
)ver^r$_{\Sigma,n}$: $2^{T_\Sigma(X_m)} \to 2^{T_\Sigma(X_m)}$ be a function such t'
:or any set $P \subseteq T_\Sigma(X_m)$,
)ver$_{\Sigma,n}$ (P) = {t | t∈P and $l_\Sigma(t) > n$}. In other wor
Ls the set of trees with more than n nodes labele
Σ.

Example 4.5.1: Let Σ = {a,f,g} where r(a)=0, r
and r(g)=1. Then,

```
overr   ({a, g,  f  ,  f  ,  f  })
     Σ,3     I  / \   / \   / \
             a  a  a g   a a   g
                     I         I
                     a         a

      - {   f  ,   f  }.
           / \    / \
          g   a  a   g
          |          |
          a          a
```

In order to use overr to test if a set of
productions is in n-normal form, one must abstrac
right hand sides of the productions. Given a tre
grammar G=(|,Σ,P,S), let rhs .: $2^P \to 2^{T_\Sigma \cup \{\ldots\}(X_m)}$ b(
function such that given any subset $P' \subseteq P$,

s(P′) = $\{t \mid F(\vec{x}) \to t \in P'\}$.

ample **4.5.2:** Let $G_1=(\Phi,\Sigma,P,S)$ such that

$\Phi = \{S,F\}$ where $r(S)=0$ and $r(F)=1$;

$\Sigma = \{a,f\}$ where $r(a)=0$ and $r(f)=1$; and

```
P = {S -> F , F -> f , F -> x }.
          |    |    |    |
          a    x    F    x
                    |
                    a
```

```
Then, rhs(P) = { F , f , x }
                 |   |
                 a   F
                     |
                     a
```

```
and rhs({S -> F , F -> x}) = { F , x }.
              |   |            |
              a   x            a
```

Combining the two previously defined functions e can formally define what it means to be in n-no rm. Given a tree grammar $G=(\Phi,\Sigma,P,S)$, <u>G is in</u> normal form, for some $n\geq 0$, if and only if $er_{\Sigma\vee\Phi,n}(rhs(P))=\emptyset$. In other words, the right-ha des of all productions in P are labeled by at mos nterminal and terminal symbols.

Example **4.5.3:** Let G_1 be defined as in example 4.5.2. Then G_1 is in 3-normal form but not 2-no form since

$$\mathrm{over}_{\bar{\Sigma} \vee \bar{\Phi}, 2}(\mathrm{rhs}(P)) = \left\{ \begin{array}{c} f \\ | \\ F \\ | \\ a \end{array} \right\}.$$

Having defined the meaning of a tree gram in n-normal form, the next step is to show how grammar in n-normal form can be converted to 2 form. Rather than accomplish this transformat one step, it is done via a series of transform such that each transformation reduces the size production, from n-normal form, to (n-1)-norma In this transformation, it is important to be pick out the leftmost subtree, of the root, wh not a variable. Having located that node, the will combine the root and the leftmost immedia descendant node of the root, not labeled with variable, into a single node. Hence, if the p production was in n-normal form, the transform production is in (n-1)-normal form. To assist transformation, the following definitions are

Definition 4.5.1: Given a ranked alphabet $\bar{\Sigma}$, a $t \in T_{\bar{\Sigma}}(X_m)$ where $l_{\bar{\Sigma}}(t) > 2$ and some $i \in \mathbf{N}_+$ such that

i) $i \in dom(t)$ and $t(i) \in \bar{\Sigma}$

ii) for all j, $1 \leq j < i$, $t(j) \notin \bar{\Sigma}$

en i is the *the leftmost nonvariable descendant* (

n) *of the root*, denoted as ls(t).

Example 4.5.4: Let $\bar{\Sigma} = \{a,f,g\}$ where $r(a)=0$, $r(f$

and $r(g)=1$. Then

```
ls(   f   ) = 1,  ls(   f   ) = 2,  ls( g ) = 1,
     / \               / \              |
    g   a             x   g             a
    |                     |
    a                     a

and ls( g ) is not defined since l_Σ( g ) = 1.
        |                            |
        x                            x
```

finition 4.5.2: Given a tree grammar $G=(\Phi,\bar{\Sigma},P,S)$,

oduction $F(\bar{x}) \to t \in P$ such that $l_{\bar{\Sigma}\vee\Gamma}(t)>2$, then the

duced nonterminal of t, denoted as NT(t), is a ne

phabet symbol T such that

i) $T \notin \bar{\Sigma} \vee \Phi$

ii) $r(T) = r(t(\varepsilon))+r(t(ls(t)))-1$

other words, NT(t) is a new nonterminal which wi

used to replace the root and node ls(t) of the

oduction's right-hand side with a single node lab

th T.

<u>**Definition**</u> <u>**4.5.3:**</u> Given a context-free tree gramm

$G=(\Phi,\Sigma,P,S)$, a production $F(\vec{x})\text{->}t\in P$ such that

$L_{\Sigma\cup\bar{\Phi}}(t)>2$, the <u>simplified right hand side of $F(\vec{x})$</u>

denoted as $S_{rhs}(G,F(\vec{x})\text{->}t)$, is the tree

$NT(t)(t/1,\ldots,t/i-1,t/i\cdot 1,\ldots,t/i\cdot j,t/i+1,\ldots,t/q$

where $i=ls(t)$, $j=r(t(ls(t)))$, and $q=r(NT(t))$. Th

<u>Nonterminal expansion of $F(\vec{x})\text{->}t$</u>, denoted

$E_{NT}(G,F(\vec{x})\text{->}t)$, is the tree whose graph is the se

pairs $\{(\varepsilon,t(\varepsilon)),(ls(t),t(ls(t)))\}\cup$

$\{(i,x_i)\mid 1\leq i<ls(t)\}\cup$

$\{(ls(t)\cdot j,x_{ls(t)+j-1})\mid 1\leq j\leq r(t(ls(t)))\}\cup$

$\{(i,x_{i+ls(t)+r(t(ls(t)))}\mid 1\leq i<r(t(\varepsilon))\}$

<u>**Example**</u> <u>**4.5.5:**</u> Let G_1 be defined as in example 4.5.2.

Clearly $\begin{array}{c}F\\|\\x\end{array}\ \text{->}\ \begin{array}{c}f\\|\\F\\|\\a\end{array}$ is a production such that

$L_{\Sigma\cup\bar{\Phi}}\left(\begin{array}{c}f\\|\\F\\|\\a\end{array}\right)>2$. Let $NT\left(\begin{array}{c}f\\|\\F\\|\\a\end{array}\right)=T$ where $r(T)=1$.

Then, $S_{rhs}\left(G_1,\ \begin{array}{c}F\\|\\x\end{array}\ \text{->}\ \begin{array}{c}f\\|\\F\\|\\a\end{array}\right)=\begin{array}{c}T\\|\\a\end{array}$

$$E_{NT}\left(G_1\ ,\ \begin{array}{c} F\\ |\\ x \end{array} \to \begin{array}{c} f\\ |\\ F\\ |\\ a \end{array}\right) = \begin{array}{c} f\\ |\\ F\\ |\\ x \end{array} .$$

aving defined the above, the method to transform grammar to 2-normal form can be introduced. One note that the transformation is an iterative s where on each iteration, the transformation tes a tree grammar which is closer to 2-normal and the iteration process terminates after a number of iterations.

iven a tree grammar $G_1=(\Phi_1,\Sigma,P_1,S_1)$ in n-normal for some $n>2$, and any production $F(\vec{x})\to t\in P_1$ such $\in cover_{\Sigma\cup\Phi_1,n-1}(rhs(P_1))$, let $G_2=(\Phi_2,\Sigma,P_2,S_2)$ be <u>reduced grammar of G_1 using $F(\vec{x})\to t$</u>, denoted $d_n(G_1,F(\vec{x})\to t)$, such that:

) $\Phi_2=\Phi_1\cup\{S_2,T\}$ such that $S_2,T\notin\Phi_1\cup\Sigma$, $S_2\neq T$

$r(S_2)=r(S_1)=0$ and $T=NT(t)$.

) $P_2 = (P_1-\{F(\vec{x})\to t\}) \cup \{S_2\to S_1,$

$F(\vec{x})\to S_{rhs}(G_1,F(\vec{x})\to t),\ T(\vec{x})\to E_{NT}(G_1,F(\vec{x})\to t)$

Example 4.5.6: Let G_1 be defined as in example 4.5.2. Then,

```
G2 = reduced3(G1 , F -> f) is the
                   |    |
                   x    F
                        |
                        a
```

tree grammar such that

Φ_2 = {S_2,S,F,T} where r(S_2)=r(S)=0 and r

Σ = {a,f} where r(a)=0 and r(f)=1; and

```
P2 = {S2 -> S , S -> F , F -> T ,
                     |    |    |
                     x    x    a

       T -> f , F -> x }.
       |    |   |
       x    F   x
            |
            x
```

To show that this transformation conve grammar to an equivalent tree grammar, the four lemmas are presented:

Lemma 4.5.1: Given a tree grammar G_1=(Φ_1,Σ, n-normal form, for any n>2, a production F(such that t$\in$over$_{\Sigma\vee\Phi_1,n-1}$(rhs(P_1)), and G_2=reduced$_n$(G_1,F($\vec{x}$)->t), then

$F(\vec{x}) \xRightarrow[OI]{}_{G_2} S_{rhs}(G_1,F(\vec{x})\text{->}t) \xRightarrow[OI]{}_{G_2} t.$

The proof of the above lemma is left as an
xercise for the reader.

emma **4.5.2:** Given a tree grammar $G_1=(\Phi_1,\Sigma,P_1,S_1)$
-normal form, for any $k>2$, a production $F(\vec{x})\to t\in P$
uch that $t\in over_{\Sigma\vee\Phi_1,k-1}(rhs(P_1))$, and
$_2$=reduced$_k(G_1,F(\vec{x})\to t)$, if $S_1 \underset{OI}{\Longrightarrow}{}^{n}_{G_1} t_1$, then
$_2 \underset{OI}{\Longrightarrow}{}^{*}_{G_2} t_1$.

roof: By Induction on n.

ase case: $S_1 \underset{OI}{\Longrightarrow}{}^{0}_{G_1} t_1$. By the definition of G_2,
learly $S_2 \Longrightarrow_{G'} S_1 = t_1$.

nductive step: Assume $S_1 \underset{OI}{\Longrightarrow}{}^{n}_{G_1} t_1 \underset{OI}{\Longrightarrow}_{G_1} t_2$ where
$_1=s[u \leftarrow G(s_1,\ldots,s_q)]$, $t_2=s[u \leftarrow t'(s_1,\ldots,s_q)]$
hat $G(\vec{x})\to t'\in P_1$ and $r(G)=q$. By induction, clearl
$_2 \underset{OI}{\Longrightarrow}{}^{*}_{G_2} t_1$. If $G(\vec{x})\to t'\neq F(\vec{x})\to t$, then by the
efinition of G_2, $G(\vec{x})\to t'\in P_2$. But then $t_1 \underset{OI}{\Longrightarrow}_{G_2}$
n the other hand if $G(\vec{x})\to t'=F(\vec{x})\to t$, then by lem
.5.1, $t_1=s[u \leftarrow G(s_1,\ldots,s_q)] \underset{OI}{\Longrightarrow}{}^{2}_{G_2}$
$[u \leftarrow t(s_1,\ldots,s_q)]=t_2$. Hence $S_2 \underset{OI}{\Longrightarrow}{}^{*}_{G_2} t_2$.

emma **4.5.3:** Given a tree grammar $G_1=(\Phi_1,\Sigma,P_1,S_1)$
-normal form, for some $k>2$, a production $F(\vec{x})\to t$
uch that $t\in over_{\Sigma\vee\Phi_1,k-1}(rhs(P_1))$,
$_2$=reduced$_k(G_1,F(\vec{x})\to t)$, and any two trees
$_1,t_2\in T_{\Sigma\vee\Phi_1}(X_A)$, if $t_1 \underset{OI}{\Longrightarrow}{}^{n}_{G_2} t_2$, then $t_1 \underset{OI}{\Longrightarrow}{}^{*}_{G_1} t_2$

roof: By induction on n.

ase case: n=0 - Trivial.

nductive step: Assume $t_1 \overset{n}{\underset{OI}{\Longrightarrow}}_{G_2} t_3 \underset{OI}{\Longrightarrow}_{G_2} t_2$ such
3 $\underset{OI}{\Longrightarrow}_{G_2} t_2$ using $G(\vec{x})\text{->}t'$. Inspecting the defini
f G_2, there are two cases depending on whether or
$(\vec{x})\text{->}t' \in P_1$.

ase 1: $G(\vec{x})\text{->}t' \in P_1$. Hence $t_3 = s[u \leftarrow G(s_1,\ldots,s_q$
$\overline{I}\!>_{G_1}$ $s[u \leftarrow t'(s_1,\ldots,s_q)] = t_2$. Since $t_2 \in T_{\overline{\Sigma} \vee \overline{\Phi}_1}(\mathbf{X}_m$
nd $G \in \overline{\Phi}_1$, clearly $t_3 \in T_{\overline{\Sigma} \vee \overline{\Phi}_1}(\mathbf{X}_m)$. Hence, by inducti
1 $\overset{*}{\underset{OI}{\Longrightarrow}}_{G_1} t_2$.
ase 2: $G(\vec{x})\text{->}t' \notin P_1$. By the definition of G_2, th
re three possibilities:

i) $G(\vec{x})\text{->}t' = S_2\text{->}S_1$

ii) $G(\vec{x})\text{->}t' = F(\vec{x})\text{->}S_{rhs}(G_1, F(\vec{x})\text{->}t)$

iii) $G(\vec{x})\text{->}t' = T(\vec{x})\text{->}E_{NT}(G_1, F(\vec{x})\text{->}t)$ where
$T = NT(t)$.

owever, since $t_2 \in T_{\overline{\Sigma} \vee \overline{\Phi}_1}(\mathbf{X}_m)$, clearly only conditio
ii) can apply. Furthermore, since $T \notin \overline{\Phi}_1$, clearly
3 $\notin T_{\overline{\Sigma} \vee \overline{\Phi}_1}(\mathbf{X}_m)$, and hence it must be the case that
1 $\overset{n-1}{\underset{OI}{\Longrightarrow}}_{G_2} t_4 \underset{OI}{\Longrightarrow}_{G_2} t_3 \underset{OI}{\Longrightarrow}_{G_2} t_2$. By inspection of
ight hand sides of the productions in P_2, for
ccurrences of T, it must be the case that $t_4 \underset{OI}{\Longrightarrow}_G$
sing $F(\vec{x})\text{->}S_{rhs}(G_1, F(\vec{x})\text{->}t)$. But then

$t_4 = s[u \leftarrow F(s_1,\ldots,s_q)] \overrightarrow{OI} G_2$

$s[u \leftarrow T(t/1,\ldots,t/i-1,t/i\cdot 1,\ldots,t/i\cdot j,t/i+1,$

$\ldots,t/q')(s_1,\ldots,s_q)] = t_3,\ t_3 \overrightarrow{OI} G_2$

$s[u \leftarrow t(\epsilon)(t/1,\ldots,t/i-1,t(t/i\cdot 1,\ldots,t/i\cdot j),t/i+$

$\ldots,t/q')(s_1,\ldots,s_q)] = s[u \leftarrow t(s_1,\ldots,s_q)] = t$

where u€dom(t4), i*ls(t), j=r(ls(t)), and q'=*t(t(

Since t_0€T-r-w (X), clearly t.ST^,- (X) and hen

by induction^ t_1 ^y>G_1 t_4. Also, using F(lf)->t€P

$t_4 = s[u \leftarrow F(s_1,\ldots,s_q)] \Rightarrow_{G_1} s[u \leftarrow t(s_1,\ldots,s_q$

Hence t_x OI > ^ t^

Lemma 4.5.4; Given a tree grammar G.^CJI 1>i.,P1 >S1)

n-normal form, for any n>2, a production F(lt)->t€

such that t6overr-ws f(rhs(Pt))f and

G^reduced (G , F(lt)->t) , then L (G) - L (G).

Proof; By the definition of a tree language,

L^TCG) - \t 1 S =T>> t and t€T-r-} • By lemma 4t

if St T-T^,, t where t€T^-, then So Ki>r. t. Hence

LnT(G1) ^ LnT(G^). On the other hand, for any t^

such that S^ Wj^n t, One wants to show that

Si ^r>n t- Clearly, since $S_0 \rightarrow S_1$ is the only

production in G« with S^ on its left hand side,

S2 7ry>G2 S1 =•> G2^ t. By lemma 4.5.3, since

Sl ^T>G2 tj Sl ^1^*0 1 t# But thetl L0I^G2^ - L0I^C

Hence $L_{OI}(G$^ - $L_{OI}(G_2)$.

The following lemma shows that each transformat
uces the number of productions which are in n-nor
m but not (n-1)-normal form.

ma **4.5.5:** Given a tree grammar $G_1=(\bar{\Phi}_1,\bar{\Sigma},P_1,S_1)$ in
ormal form, for any n>2, such that there exists a
duction $F(\vec{x})\to t\in P_1$ such that
$\mathrm{ver}_{\bar{\Sigma}\vee\bar{\Phi}_1,n-1}(\mathrm{rhs}(P_1))$, and $G_2=\mathrm{reduced}_n(G_1,F(\vec{x})\to t$
n $|\mathrm{over}_{\bar{\Sigma}\vee\bar{\Phi}_1,n-1}(\mathrm{rhs}(P_1))| =$
$\mathrm{er}_{\bar{\Sigma}\vee\bar{\Phi}_2,n-1}(\mathrm{rhs}(P_2))| + 1$.

of: By inspecting the definitions,

$\mathrm{r}_{\bar{\Sigma}\vee\bar{\Phi}_2,n-1}(\mathrm{rhs}(P_2)) =$
$\mathrm{r}_{\bar{\Sigma}\vee\bar{\Phi}_2,n-1}(\mathrm{rhs}(P_1-\{F(\vec{x})\to t)\})) \vee$
$\mathrm{r}_{\bar{\Sigma}\vee\bar{\Phi}_2,n-1}(\{S_2,S_{\mathrm{rhs}}(G_1,F(\vec{x})\to t),E_{NT}(G_1,F(\vec{x})\to t)\})$
ce for all $t\in\mathrm{rhs}(P)$, $t\in T_{\bar{\Sigma}\vee\bar{\Phi}_1}(X_A)$, and
$\bar{\Phi}_2=\bar{\Sigma}\vee\bar{\Phi}_1\vee\{S_2,T\}$, clearly
$\mathrm{r}_{\bar{\Sigma}\vee\bar{\Phi}_2,n-1}(\mathrm{rhs}(P_1-\{F(\vec{x})\to t\})) =$
$\mathrm{r}_{\bar{\Sigma}\vee\bar{\Phi}_1,n-1}(\mathrm{rhs}(P_1))-\{t\}$. Since G_1 is in n-normal
m, for any tree $t'\in\mathrm{over}_{\bar{\Sigma}\vee\bar{\Phi}_1,n-1}(\mathrm{rhs}(P_1))$,
$/_{\bar{\Phi}_1}(t')=n$. Hence, for the production $F(\vec{x})\to t$,
$/_{\bar{\Phi}_1}(t)=n$. By inspection of $F(\vec{x})\to S_{\mathrm{rhs}}(G_1,F(\vec{x})\to t$
$/_{\bar{\Phi}_2}(S_{\mathrm{rhs}}(G_1,F(\vec{x})\to t))=n-1$, since the labeled node
e been reduced by one. Similarly,

$/_{\bar{\Phi}_2}(T(\vec{x})\to E_{NT}(G_1,F(\vec{x})\to t))=2$, and $1_{\bar{\Sigma}\vee\bar{\Phi}_2}(S_2)=1$.
n $\mathrm{over}_{\bar{\Sigma}\vee\bar{\Phi}_2,n-1}(\{S_2,S_{\mathrm{rhs}}(G_1,F(\vec{x})\to t)$,

$(G_1,F(\vec{x})\text{->}t)\})=\emptyset$. Hence $over_{\overline{\Sigma}\vee\overline{\Phi}_2,n-1}(rhs(P_2)) =$ $over_{\overline{\Sigma}\vee\overline{\Phi}_2,n-1}(rhs(P_1))-\{t\}$, or

$|over_{\overline{\Sigma}\vee\overline{\Phi}_1,n-1}(rhs(P_1))|=|over_{\overline{\Sigma}\vee\overline{\Phi}_2,n-1}(rhs(P_2))| + 1$

Using the above lemmas, one can show that there [is a f]inite sequence of transformations such that a tre[e gra]mmar, in n-normal form, can be reduced to 2-norma[l for]m. This is shown by the following theorem:

<u>[The]orem</u> <u>4.5.1:</u> Given a tree grammar $G_0=(\Phi_0,\overline{\Sigma},P_0,S_0)$ [in n-n]ormal form, for any n>2, there exists a finite [seq]uence of tree grammars $G_0,G_1,\ldots,G_q$ such that G_q [is in] (n-1)-normal form where:

i) $G_i = (\Phi_i,\overline{\Sigma},P_i,S_i)$ for all i, $0\leq i\leq q$;

ii) $G_i = reduced_n(G_{i-1},p_{i-1})$ where $p_{i-1}\in P_{i-1}$ is [a] production of the form $F(\vec{x})\text{->}t$ such that $t\in over_{\overline{\Sigma}\vee\overline{\Phi}_{i-1},n-1}(rhs(P_{i-1}))$ for all i, $1\leq i\leq$[q];

iii) $L_{OI}(G_i) = L_{OI}(G_{i-1})$ for all i, $1\leq i\leq q$; and

iv) $q = |over_{\overline{\Sigma}\vee\overline{\Phi}_0,n-1}(rhs(P_0))|$.

roof: By induction on q.

ase step: $q = 0$. By the definition of n-normal orms, G_0 is in (n-1)-normal form since $\mathrm{over}_{\Sigma \vee \Phi_0, n-1}(\mathrm{rhs}(P_0)) = \emptyset$.

nductive step: Assume $q = k+1$ for some $k>0$. By .5.4, there exists a tree grammar $_1 = \mathrm{reduced}_n(G_0, F(\vec{x}) \to t)$, where $F(\vec{x}) \to t \in P_0$ and $\in \mathrm{over}_{\Sigma_0 \vee \Phi_0, n-1}(\mathrm{rhs}(P_0))$, such that $L_{OI}(G_0) = L_{OI}($ y lemma 4.5.5, $|\mathrm{over}_{\Sigma_1 \vee \Phi_1, n-1}(\mathrm{rhs}(P_1))| = q-1$. ence, if q is finite, then by induction, there ex finite sequence of tree grammars $G_1, \ldots, G_q$ such onditions i) through iv) of the theorem are met. hen the sequence of tree grammars, for the k+1 ca s simply $G_0, G_1, \ldots, G_q$. By inspection of the efinition of over, $\mathrm{over}_{\Sigma \vee \Phi_0, n-1}(\mathrm{rhs}(P_0)) \subseteq P_0$. B he definition of a tree grammar P_0 is finite. He ust be finite and hence $G_0, G_1, \ldots, G_q$ exists meeti he conditions of the theorem.

orollary: Given a context-free tree grammar $_0 = (\Phi_0, \Sigma, P_0, S_0)$ in n-normal form, for any $n \geq 2$, the xists a finite sequence of context-free tree gram $_0, \ldots, G_{q'}$ such that $G_{q'}$ in in 2-normal form and f ll i, $1 \leq i \leq q'$, $L_{OI}(G_i) = L_{OI}(G_{i-1})$.

.6 Derivation-renaming Grammars

The objects of study, in this section, are t grammars containing productions for which the rig and side of some production is a tree where the s labeled with a nonterminal symbol, and all oth odes in the tree are labeled with variables. Mo ormally, given a tree grammar $G=(\Phi,\Sigma,P,S)$, let t of trees with just the root labeled by a nontermi lenoted $SN(\Phi)$, be the set $\{t \mid t \in T_{\Phi}(X_m)$ and $1_{\Phi}(t)$ lence, a production $F(\vec{x})\text{->}t \in P$ is <u>derivation</u> <u>renam</u> nd only if $t \in SN(\Phi)$. Similarly, a tree grammar G <u>lerivation</u> <u>renaming</u> <u>grammar</u> if and only if there a production p such that p is derivation renaming Note that a production $F(\vec{x})\text{->}t$ is called derivati renaming, since if $F(\vec{x})\text{->}t$ is used in a derivatio step, the net effect is to rename the nonterminal labelling a given node, of the derived tree, with another nonterminal symbol (and possibly trim off and/or duplicate some of it's subtrees).

Example 4.6.1: Let $G_1=(\Phi,\Sigma,P,S)$ such that

Φ = {S,G,F} where r(S)=0 and r(G)=r(F)=1;

Σ = {a,f} where r(a)=0 and r(f)=1; and

$$P = \{ S \to \begin{array}{c} G \\ | \\ a \end{array},\ \begin{array}{c} G \\ | \\ x \end{array} \to \begin{array}{c} F \\ | \\ x \end{array},\ \begin{array}{c} F \\ | \\ x \end{array} \to \begin{array}{c} f \\ | \\ a \end{array} \}.$$

Then, G_1 is derivation renaming since

$$\begin{array}{c} G \\ | \\ x \end{array} \to \begin{array}{c} F \\ | \\ x \end{array} \text{ is derivation renaming.}$$

The intent of this section is to show t derivation renaming tree grammar G_1 can be t to a tree grammar G_2 such that G_2 is derivat renaming free and $L_{OI}(G_1)=L_{OI}(G_2)$. One shou that the notion of eliminating derivation re productions parallels the notion of eliminat "chain-rules" in string grammars (see Harris Bar-Hillel, Perles, and Shamir[61]) where an set of nonterminals is created such that eac nonterminal in the set can be reached by a c

Like chain rules in string grammars, th transformation uses an inductive set of nont where each nonterminal in the set can be rea chain-rule. Given a tree grammar $G=(\Phi,\Sigma,P,S$ nonterminals $F\in\Phi$, let $I_F = \{t_n \mid F(\vec{x}) \underset{OI}{\Longrightarrow} t_1$ $\underset{OI}{\Longrightarrow} \cdots \underset{OI}{\Longrightarrow} t_n$ for all $n\geq 1$, and for all i, $t_i \in SN(\Phi)\}$.

To find each I_F effectively, let I_F be inducti
:ined as follows:

i) Let $I_{F,1} = \{t \mid t \in SN(\Phi)$ and $F(\vec{x}) \to t \in P\}$

ii) For any $n \geq 1$, let $I_{F,n+1} = I_{F,n} \cup \{t \mid t \in SN$
and there exists a tree $t' \in I_{F,n}$ such that
$t' \overset{}{\underset{OI}{\Longrightarrow}} t$ using some production $G(\vec{x}) \to s \in P$ w
$s \in SN(\Phi)\}$

ıce P and Φ are finite, it is clear that for ever
l, and every $F \in \Phi$, $I_{F,n}$ can be constructed. By th
iuctive definition of $I_{F,n}$, clearly
$,1 \subseteq I_{F,2} \subseteq \ldots \subseteq SN(\Phi)$. Hence, for each $F \in \Phi$, th
ists a least k_F such that

i) $k_F \leq |SN(\Phi)|$

ii) For all $n \geq k_F$, $I_{F,n} = I_{F,n+1}$.

ıce, for any $j \geq 1$, $I_{F,k_F+j} = I_{F,k_F}$. One would lik
ow that in fact $I_{F,k_F} = I_F$, which is shown by the
llowing three lemmas.

mma 4.6.1: Given any tree grammar $G=(\Phi,\Sigma,P,S)$, an
l, any $F \in \Phi$, $I_{F,n} \subseteq I_F$.

Proof: By induction on n.

base case: Let $t \in I_{F,1}$. By the definition of I_F, there exists a production $F(\vec{x}) \rightarrow t \in P$ where $t \in SN(\bar{\Phi})$ then $F(\vec{x}) \overset{\longrightarrow}{\text{OI}} t$ and hence, $I_{F,1} \subseteq I_F$.

Inductive step: Let $t \in I_{F,n+1}$. By the definition $I_{F,n+1}$, either $t \in I_{F,n}$ or $t \in (I_{F,n+1} - I_{F,n})$. If $t \in I$ by induction, $t \in I_F$. If $t \in (I_{F,n+1} - I_{F,n})$, then by definition, there exists a $t_n \in I_{F,n}$ and $t_n \overset{\longrightarrow}{\text{OI}} t$ u $G(\vec{x}) \rightarrow s \in P$ where $s, t \in SN(\bar{\Phi})$. By induction we know $I_{F,n} \subseteq I_F$, and hence $F(\vec{x}) \overset{\longrightarrow}{\text{OI}} t_1 \overset{\longrightarrow}{\text{OI}} t_2 \overset{\longrightarrow}{\text{OI}} \ldots \overset{\longrightarrow}{\text{O}}$ such that for all i, $1 \leq i \leq n$, $t_i \in I_F$. But then, by definition, $t \in I_F$. Hence, $I_{F,n+1} \subseteq I_F$.

Lemma 4.6.2: Given any tree grammar $G=(\bar{\Phi},\Sigma,P,S)$, $F \in \bar{\Phi}$, $I_F \subseteq I_{F,k_F}$.

Proof: Assume $t_n \in I_F$ such that $t_n \notin I_{F,k_F}$. By the definition of I_F, $F(\vec{x}) \overset{\longrightarrow}{\text{OI}} t_1 \overset{\longrightarrow}{\text{OI}} \ldots \overset{\longrightarrow}{\text{OI}} t_n$ wher each i, $1 \leq i \leq n$, $t_i \in SN(\bar{\Phi})$. Since t_1 is obtained in derivation step, it must be the case that $F(\vec{x}) \rightarrow t$ and hence $t_1 \in I_{F,1}$. More generally, for any j, $1 \leq$ $t_j \in I_{F,j}$ and t_j must be of the form $G(x_1',\ldots,x_q')$ w $G \in \bar{\Phi}$, $q=r(G)$, and $x_1',\ldots,x_q' \in \mathbf{X}_{r(F)}$. If $t_j \overset{\longrightarrow}{\text{OI}} t_{j+1}$ the definition of a derivation, there must exist production $G(\vec{x}) \rightarrow s \in P$ such that $t_{j+1} = s(x_1',\ldots,x_q')$. Assume $s \notin SN(\bar{\Phi})$. Clearly $1_{\Sigma \vee \bar{\Phi}}(s(x_1',\ldots,x_q')) \neq 1$ whi contradiction since $t_{j+1} \in SN(\bar{\Phi})$. Hence $s \in SN(\bar{\Phi})$. hen by definition, $t_{j+1} \in I_{F,n+1}$. Also, since k_F i east value such that for all $n \geq k_F$, $I_{F,n} = I_{F,n+1}$, learly $I_F \subseteq I_{F,k_F}$.

emma 4.6.3: Given any tree grammar $G=(\bar{\Phi},\bar{\Sigma},P,S)$, a $\in\bar{\Phi}$, $I_{F,k_F} = I_F$.

roof: This follows directly from lemmas 4.6.1 anc .6.2.

The next step is to use I_F to convert a tree rammar G_1 to an equivalent tree grammar G_2 (under I derivation) which removes productions of the fo $(\vec{x})\text{->}t$ where $t \in SN(\bar{\Phi})$.

Given a tree grammar $G_1=(\bar{\Phi},\bar{\Sigma},P_1,S)$, let $_2=(\bar{\Phi},\bar{\Sigma},P_2,S)$, called the <u>derivation renaming free</u> <u>rammar of G_1</u> and denoted $drf(G_1)$, such that

$_2 = (P_1-\{F(\vec{x})\text{->}t \in P_1 \mid t \in SN(\bar{\Phi})\} \vee \{F(\vec{x})\text{->}t \mid t' \in I$

here $t' \overset{}{\underset{OI}{\Longrightarrow}} t$ using $G(\vec{x})\text{->}s \in P_1$ and $s \notin SN(\bar{\Phi})\}$

here for each $F \in \bar{\Phi}$, I_F is defined using the tree rammar G_1.

Example 4.6.2: Let $G_1=(\bar{\Phi},\bar{\Sigma},P_1,S)$ such that

$\bar{\Phi} = \{S,F,G,H\}$ where $r(s)=0$, $r(F)=1$, and $r(G)=r$

$\bar{\Sigma} = \{a,f\}$ where $r(a)=0$ and $r(f)=1$; and

```
P  => { S -> F , F ->  G  ,  G  ->  H  ,
 1          II        / \      / \      / \
            a     x     x     x x    y   y     x

                   H  -> F ,  H  -> f }.
                  / \     I    / \     I
                 x   y    x   x   y    x

Then, I  - 0, I  - { G  ,  H  , F },
       S       F     / \   / \   I
                    x   xx       xx

I. = { H  , F ,  G  ,  H  } and
 G    / \   I   / \   / \
     y   x  y   y   y y   y

I  - { F ,  G  ,  H  }.
 H     I   / \   / A
       xx    xx    x
```

Furthermore, G^^drfCG$_1$) is the tree grammar

G$_2$=(1,!,P$_2$,S) where

```
P2 - { S -> F ,  H  -> f ,  F -> f,  G  -> f
            I   / \   I    I    I  / \   I
            a x   y   x    x    xx  y    y
```

The next theorem and two lemmas show that

two tree grammars G_1 and G_2 such that G_2=drf(G

$L_{OI}(G_1) - L_{OI}(G_2)$-

__Lemma__ __4.6.4;__ Given a tree grammar G_1»(J,$\bar{I}$,P_{1f}S

G_2=drf(G_1), $t_1 \in T_{\Sigma \vee \Phi}(X_A)$, and $t_2 \in T_p$ if $t_1 \overset{*}{\underset{OI}{\Longrightarrow}}$

$t_1 \overset{*}{\underset{OI}{\Longrightarrow}}_{G_2} t_2$.

roof: By induction on n.

ase case: n=o. Trivial.

nductive step: $t_1 \underset{OI}{\Longrightarrow}_{G_1} t_3 \underset{OI}{\Longrightarrow}{}^{n}_{G_1} t_2$ where

$_1=s[u \leftarrow F(s_1,\ldots,s_q)]$, $t_2=s[u \leftarrow t(s_1,\ldots,s_q)]$,

$(\vec{x})\rightarrow t\in P_1$ and $r(F)=q$. Depending on whether or n

$(\vec{x})\rightarrow t\in P_2$, there are two cases:

ase 1: $F(\vec{x})\rightarrow t\in P_2$. Hence $t_1 \underset{OI}{\Longrightarrow}_{G_2} t_3$. By induc

$_3 \underset{OI}{\Longrightarrow}{}^{*}_{G_2} t_2$ and hence $t_1 \underset{OI}{\Longrightarrow}{}^{*}_{G_2} t_2$.

ase 2: $F(\vec{x})\rightarrow t\notin P_2$. Hence, by the definition of

$\in SN(\Phi)$. Since $t_2\in T_\Sigma$, it must be the case that

$_1 \underset{OI}{\Longrightarrow}_{G_1} t_3 \underset{OI}{\Longrightarrow}_{G_1} t_4 \underset{OI}{\Longrightarrow}_{G_1} \cdots \underset{OI}{\Longrightarrow}_{G_1} t_{j+2} \underset{OI}{\Longrightarrow}_{G_1} t$

$\underset{I}{\Longrightarrow}{}^{m}_{G_1} t_2$ where

i) $j+m+1 = n$ where $j>0$,

ii) for all i, $1\leq i<j$, $t_{i+2} \underset{OI}{\Longrightarrow}_{G_1} t_{i+3}$ using s production of the form $G(\vec{x})\rightarrow t'\in P_1$ such t $t'\in SN(\Phi)$,

iii) $t_{j+2} \underset{OI}{\Longrightarrow}_{G_1} t_{j+3}$ using some production $H(\vec{x})\rightarrow t''\in P_1$ where $t''\notin SN(\Phi)$.

y the definition of I_F, $t_{j+2}\in I_F$. Hence, $F(\vec{x})\rightarrow t''$

ut then $t_1 \underset{OI}{\Longrightarrow}_{G_2} t_{j+3}$. Since $m=n-(j+1)$, clearly

y induction $t_{j+3} \underset{OI}{\Longrightarrow}{}^{*}_{G_2} t_2$. Hence $t_1 \underset{OI}{\Longrightarrow}{}^{*}_{G_2} t_2$.

__Lemma__ __4.6.5:__ Given a tree grammar $G_1=(\Phi,\Sigma,$

$G_2=drf(G_1)$, if $S \overset{n}{\underset{OI}{\Longrightarrow}}_{G_2} t$, then $S \overset{*}{\underset{OI}{\Longrightarrow}}_{G_1} t$.

__Proof:__ By induction on n.

__base__ __step__: $n=0$. Trivial.

__inductive__ __step__: $S \overset{n}{\underset{OI}{\Longrightarrow}}_{G_2} t_1 \underset{OI}{\Longrightarrow}_{G_2} t_2$ where

$t_1=s[u \leftarrow F(s_1,...,s_q)]$, $t_2=s[u \leftarrow t(s_1,..$

$r(F)=q$, and $F(\vec{x}) \rightarrow t \in P_2$. By induction, S $\overline{\overline{O}}$

Depending of whether or not $F(\vec{x}) \rightarrow t \in P_1$, th

cases:

__case__ 1: $F(\vec{x}) \rightarrow t \in P_1$. Clearly $t_1 \underset{OI}{\Longrightarrow}_{G_1} t_2$ a

$S \overset{*}{\underset{OI}{\Longrightarrow}}_{G_1} t_2$.

__case__ 2: $F(\vec{x}) \rightarrow t \notin P_1$. By the definition of

must exist a $t' \in I_F$ such that $t' \underset{OI}{\Longrightarrow}_{G_1}$ t us

$G(\vec{x}) \rightarrow s \in P_1$. By the definition of I_F, it m

case that $F(\vec{x}) \underset{OI}{\Longrightarrow}_{G_1} t'_1 \underset{OI}{\Longrightarrow}_{G_1} t'_2 \underset{OI}{\Longrightarrow}_{G_1} \ldots$

But then $t_1=s[u \leftarrow F(s_1,...,s_q)] \overset{*}{\underset{OI}{\Longrightarrow}}_{G_1}$

$s[u \leftarrow t'(s_1,...,s_q)] \underset{OI}{\Longrightarrow}_{G_1} s[u \leftarrow t(s_1,..$

Hence $S \overset{*}{\underset{OI}{\Longrightarrow}}_{G_1} t_2$.

__Theorem__ __4.6.1:__ Given a tree grammar $G_1=(\Phi,$

$G_2=drf(G_1)$, then $L_{OI}(G_1)=L_{OI}(G_2)$.

__Proof:__ This follows directly from the defi

tree language, and lemmas 4.6.4 and 4.6.5.

Finally, to verify that the conversion of G_1 ... G_2, using G_2=drf(G_1), produces a tree grammar whi... not derivation renaming is the following lemma:

Lemma **4.6.6:** Given a tree grammar $G_1=(\Phi,\Sigma,P_1,S)$ a... G_2=drf(G_1), then G_2 is not a derivation renaming grammar.

Proof: Assume G_2 is a derivation renaming grammar... Hence, there exists a production $F(\vec{x})\text{->}t\in P_2$ such ... $t\in SN(\Phi)$. By the definition of P_2, $F(\vec{x})\text{->}t\in P_1$ onl... $t\in(P_1-\{H(\vec{x})\text{->}t'\notin P_1 \mid t'\in SN(\Phi)\})$ which is impossib... Hence $F(\vec{x})\text{->}t\notin P_1$ and it must be the case that the... exists a $t'\in I_F$ such that $t' \overset{}{\underset{OI}{\Longrightarrow}} t$ using some prod... $G(\vec{x})\text{->}s\in P_1$ and $s,t\notin SN(\Phi)$. But $t\in SN(\Phi)$ which is a contradiction. Hence G_2 is not a derivation rena... grammar.

4.7 **Erasing** **Grammars**

This section investigates the types of "eras... that can exist in tree grammars. One form of era... occurs when a production is an epsilon rule (i.e. nonterminal or terminal symbols occur on the righ... side of the production). A second, more subtle, of erasing occurs in "nonconservative" tree gramm... A tree grammar is considered nonconservative if t...

exists a production $p \in P$ where a variable "x" o

the left hand side of the production p but not

right hand side. Furthermore, one would like

transformation which would remove these forms

erasing. Unfortunately, the author has not di

any transformations which will remove either f

"erasing" from tree grammars, and this problem

open.

<u>**Definition 4.7.1:**</u> Given a tree grammar $G=(\Phi,\Sigma,$

production $F(\vec{x}) \to t \in P$ is an <u>epsilon rule</u> if $1_{\Sigma V}$

In other words, the right hand side must be a

tree labeled by some variable $x \in \mathbf{X}_A$. The tree

is <u>epsilon free</u> if and only if there does not

production $p \in P$ such that p is an epsilon rule.

<u>**Example 4.7.1:**</u> Let $G_1=(\Phi,\Sigma,P,S)$ be a tree g

such that

Φ = {S,F} where r(S)=0 and r(F)=1;

Σ = {a,f} where r(a)=0 and r(f)=1; and

```
P = { S -> F , F -> f , F -> x}.
           |   |    |   |
           a   x    F   x
                    |
                    x
```

G_1 is <u>not</u> epsilon free since

```
F -> x
|
```

s an epsilon rule. Furthermore, when

```
S ==> F ==> a
      |
      a
```

he node labeled with the nonterminal F is "erase

nition **4.7.2:** Given a tree grammar $G=(\Phi,\Sigma,P,S)$, luction $F(\vec{x})\to t\in P$ is considered conservative if a if for all $x\in\{x_1,\ldots,x_{r(F)}\}$, there exists a tre ress $u\in dom(t)$ such that $t(u)=x$. In other words, ables which occur on the left hand side of a luction also occur on the right hand side. larly, the tree grammar G is Conservative if and if for every production $p\in P$, p is conservative.

xample **4.7.2:** Let $G_2=(\Phi,\Sigma,P,S)$ such that

Φ = {S,F} where r(S)=0 and r(F)=3;

Σ = {a,b,f} where r(a0=r(b) and r(f)=2; and

```
 = { S -> F     ,   F   ->   f    ,   F   -> y }.
         /|\       /|\      / \      /|\
        a a a     x y z    x   z    x y z
```

$_2$ is nonconservative since the rules

```
  F  ->  f       and     F  -> y
 /|\    / \             /|\
x y z  x   z           x y z
```

re not conservative. The rule

```
  F  -> y
 /|\
x y z
```

s also an epsilon rule and hence G_1 is neither epsilon free nor conservative.

4.8 Reduced Tree Grammars

This section investigates tree grammars wh unnecessary productions in them. Tree grammars contain productions that can not be applied to sentential form, or tree grammars which contain productions that will not derive terminal trees tree grammars which contain both types of produ are the objects of study in this section. When tree grammar does not contain productions in ei these forms, the grammar is said to be reduced. a tree grammar G is considered <u>reduced</u> if and o for every production $p \in P$, there exists a deriva such that $S \xRightarrow[OI]{}^{*} t_1 \xRightarrow[OI]{} t_2$ where $t_1 \xRightarrow[OI]{} t_2$ usin production p, and $t_2 \xRightarrow[OI]{}^{*} t$ where $t \in L_{OI}(G)$. In words, the production p is used in some derivat tree in the tree language generated by the tree G.

A natural question to ask is if one can ta tree grammar G_1 which is not reduced, eliminate productions not used in any derivation, and pro tree grammar G_2 where the tree language generat is identical to the tree language generated by

ıfortunately, the author has not discovered any
:fective method which will eliminate the unnecess
:oductions. The reason of failure is due to the
:oblem introduced by "erasing" of nonconservative
:ammars. Hence, the problem of transforming tree
:ammars into reduced tree grammars will be left o

However, as shown below, this section does pr
ı effective transformation to produce a weakly re
:ee grammar. A tree grammar $G=(\Phi,\Sigma,P,S)$ is consi
<u>eakly</u> <u>reduced</u> if and only if for every production
ıere exists a derivation such that $S \overset{*}{\underset{OI}{\Longrightarrow}} t_1$ and
ı $\underset{OI}{\Longrightarrow} t_2$ using the production p. In other words,
/ery production p, there exists some sentential f
ıch that the production p can be applied to it in
[derivation.

One should note that the methods used here ar
ıalogous to those used by Harrison[78] to produce
educed string grammars. From a given string gram
ırrison inductively builds a set W which contains
et of nonterminals which are derivable from the s
ymbol and will derive terminal strings. Then, us
, the productions eliminated are those which cont
onterminal that is not in W.

As mentioned above, one problem with tree
which does not occur in the string case is that
must concern oneself about potential "erasing"
introduced by nonconservative tree grammars. F
instance, in a nonconservative tree grammar, on
have a derivation of the form $S \underset{OI}{\Longrightarrow}^{*} t_1 \underset{OI}{\Longrightarrow} t_2$

i) $t_1 = s[u \leftarrow F(s_1,\ldots,s_m)]$ where $r(F)=m$,
$F(\vec{x}) \rightarrow t \in P$, and for some i, $1 \leq i \leq m$,
$s_i = s_i[u \leftarrow G(s_1',\ldots,s_q')]$ where $r(G)=q$.

ii) $t_2 = s[u < t(s_1,\ldots,s_m)]$ where for all w
$t(w) \neq x_i$

Hence, by rewriting with the production $F(\vec{x}) \rightarrow t$
nonterminal G (in the subtree s_i) is erased. T
classify when the nonconservative form of erasi
occurs, let D_F, for all $F \in \Phi$, be the set
$D_F = \{i \mid F(\vec{x}) \rightarrow t \in P, \text{for all } v \in dom(t)\ t(v) \neq x_i\}$.

To reduce a tree grammar G, a set R of
nonterminals is built which contains every nont
which can be rewritten in some sentential form.
$R = \{F \in \Phi \mid S \underset{OI}{\Longrightarrow}^{*} t_1$ and $t_1 \underset{OI}{\Longrightarrow} t_2$ using some
production $F(\vec{x}) \rightarrow t \in P\}$. To compute R effectivel
inductively defined as follows:

i) let $R^{0} = \{S \mid S \to t \in P\}$

ii) **for any n≥M, let $R^{n} = R^{n-1} \cup \{H \in \mathcal{J} \mid F \in R^{n-1}$**
F(lt)->t€P, u€dom(t), t(u)=H, H(x)->t'€P,
$v \in N_{+}^{+}$, vi is a prefix of u, and if t(v)=G€
then $i \notin D_{G}\}$

ince both $\mathcal{J}$ and P are finite, for every n≥0, R^{n} c
onstructed. By definition, clearly $R^{0} \subseteq R^{1} \subseteq \cdots$
ence, there exists a least $k \leq |\mathcal{J}|$ such that for al
≥e, $R^{n} = R^{n+1}$. Furthermore the following four le
how that the inductively created set R^{k} is identi
o the set R«

**emma 4.8*1; Given any tree grammar G=(f,I,P,S), a
≥0, $R^{n} \subseteq R$.**

roof: By induction on n,

ase cases:

1. $R^{0} = \emptyset$. Clearly, since there does not exis
production of the form S->t€P, there does
exist a start production. Hence, no
sentential forms can be derived and R^{0}=R«

2. $\{S\}=R^0$. By the definition of R^0, S->

then $S \overset{}{\underset{OI}{\Longrightarrow}} t$ using S->t. Hence S$\in$R.

<u>inductive</u> <u>step:</u> Let $H \in R^{n+1}$. By definition, e

$H \in R^n$ or $H \in (R^{n+1}-R^n)$. If $H \in R^n$, then by inducti

On the other hand, if $H \in (R^{n+1}-R^n)$, then by def

there exists an $F \in R^n$, $F(\vec{x})\text{->}t \in P$, $t(u)=H$, $H(\vec{x})$-

and for all prefixes vi of u, if $t(u)=G \in \Phi$, the

By induction, $S \underset{OI}{\Longrightarrow}^{*} t_1 \underset{OI}{\Longrightarrow} t_2$ where

$t_1=s[v\text{<-}F(s_1,\ldots,s_m)]$, $t_2=s[v\text{<-}t(s_1,\ldots,s_m)]$,

and $t(u)=H$. Clearly, for all ancestors of t(u

t(u) is labeled with a nonterminal, there exis

production $G(\vec{x})\text{->}s' \in P$ where for some $w \in dom(s')$

$s'(w)=x_i$ where ui is a prefix of v. But then

exists some derivation such that $s[u\text{<-}t(s_1,\ldots$

$s[u\text{<-}t''(s_1,\ldots,s_m)]$ where for some $w \in dom(t'')$,

and for all proper prefixes y of w, $t''(y) \in \Sigma$.

one can perform a rewrite on H and hence H$\in$R.

<u>Lemma</u> <u>4.8.2:</u> Given any tree grammar $G=(\Phi,\Sigma,P,S$

$S \underset{OI}{\Longrightarrow}^{n} s[u\text{<-}F(s_1,\ldots,s_m)] \underset{OI}{\Longrightarrow} s[u\text{<-}t(s_1,\ldots,s_m$

$F \in R^n$.

roof: By induction on n.

ase case: n=0 - trivial.

nductive step: $S \underset{OI}{\Longrightarrow}^{n} s[u \leftarrow F(s_1,\ldots,s_m)] \underset{OI}{\Longrightarrow}$ $[u \leftarrow t(s_1,\ldots,s_m)]$ where $r(F)=m$ and $n \geq 1$. By the efinition of an IO derivation, the derivation mus f the form $S \underset{OI}{\Longrightarrow}^{n_1} t_1 \underset{OI}{\Longrightarrow} t_2 \underset{OI}{\Longrightarrow}^{n_2} t_3 \underset{OI}{\Longrightarrow} t_4$ wher $_1=s[v \leftarrow H(s'_1,\ldots,s'_q)]$, $r(H)=q$, $t_2=s[v \leftarrow t'(s'_1,\ldots,s$ $(\vec{x}) \rightarrow t' \in P$, $t'(w)=F$, $t_3=s[v \leftarrow t''(s'_1,\ldots,s'_q)]$, $' \underset{OI}{\Longrightarrow}^{n_2} t''$, $t''(z)=F$, for all i, $1 \leq i \leq m$, $_i=t''(s'_1,\ldots,s'_q)/zi$, $_4=s[v \leftarrow t''(s'_1,\ldots,s'_q)[z \leftarrow t(t''''/z1(s'_1,\ldots,s'_q), \ldots,$ $''(s'_1,\ldots,s'_q))]]$, and $n_1+n_2=n-1$. By induction, $H\in$ or all proper prefixes yi of w in dom(t'), if $'(y)=G \in \bar{\Phi}$, clearly there must have existed a produ $(\vec{x}) \rightarrow s' \in P$ such that for some $d \in dom(s')$ $s'(d)=x_i$. he $i \notin D_G$. Hence, by definition, $H \in R^n$.

emma 4.8.3: Given any tree grammar $G=(\bar{\Phi},\Sigma,P,S)$, R

roof: Assume $F \in R$ such that $F \notin R^k$. By definition o here exists a derivation such that $S \underset{OI}{\Longrightarrow}^{n} t_1 \underset{OI}{\Longrightarrow}$ here $t_1=s[u \leftarrow F(s_1,\ldots,s_m)]$ and $t_2=s[u \leftarrow t(s_1,\ldots,s$ y lemma 4.8.2, $F \in R^n$. Clearly $R^n \subseteq R^k$ since for a $\geq k$, $R^n = R^{n+1}$. Hence $R \subseteq R^k$.

Lemma 4.8.4: Given any tree grammar $G=(\bar{\Phi},\bar{\Sigma},P$

Proof: This follows directly form lemmas 4.8
4.8.3.

Example 4.8.1: Let $G_1=(\bar{\Phi},\bar{\Sigma},P,S)$ such that

$\bar{\Phi}$ = {S,F,G,H} where r(S)=0, r(F)=r(G)=1,

$\bar{\Sigma}$ = {a,f} where r(a)=0 and r(f)=1;

```
P = {S->F, F->f, G->F,  H ->a}.
       |   |  |   |  |  / \
       G   x  x   x  x  x y
       |
       a
```

Then, R^0={S}, R^1={S,F,G}, and R={S,F,G}.

Using R, the transformation of a nonred
grammar G_1 to a weakly reduced tree grammar
defined. Given a tree grammar $G_1=(\bar{\Phi},\bar{\Sigma},P_1,S)$
$G_2=(\bar{\Phi},\bar{\Sigma},P_2,S)$ be the weakly reduced tree gra
(denoted $wr(G_1)$) such that $P_2=(P_1-\{F(\vec{x})\text{->}t\in P$
where R is defined on G_1.

Example 4.8.2: Let G_1 be defined as in ex
and $G_2=wr(G_1)$. Then

```
P2 = {S->F, F->f, G->F}
         |   |  |  |   |
         G   x  x  x   x
         |
         a
```

The following theorem and lemma show that for a

e grammar G_1, $L_{OI}(G_1) = L_{OI}(wr(G_1))$.

<u>ma</u> <u>4.8.5:</u> Given any two tree grammars G_1 and G_2

re $G_1=(\Phi,\Sigma,P_1,S)$ and $G_2=wr(G_1)=(\Phi,\Sigma,P_2,S)$, if

$\underset{OI}{\Rightarrow}{}^*_{G_1}$ t then $S \underset{OI}{\Rightarrow}{}^*_{G_2}$ t.

<u>of:</u> By induction on n.

<u>e</u> <u>case:</u> n=0. Trivial.

<u>uctive</u> <u>step:</u> $S \underset{OI}{\Rightarrow}{}^n_{G_1} t_1 \underset{OI}{\Rightarrow}_{G_1} t_2$. By induction,

$\underset{OI}{\Rightarrow}{}^n_{G_2} t_1$. Furthermore, if $t_1 \underset{OI}{\Rightarrow}_{G_1} t_2$ using

$)\text{->}t \in P_1$, then by the definition of R, $F \in R$. But

n, by the definition of P_2, $F(\vec{x})\text{->}t \in P_2$. Hence

$\underset{OI}{\Rightarrow}_{G_2} t_2$.

<u>orem</u> <u>4.8.1:</u> Given any two tree grammars

(Φ,Σ,P_1,S), and $G_2=wr(G_1)=(\Phi,\Sigma,P_2,S)$,

$(G_1) = L_{OI}(G_2)$.

Proof: By the definition of a tree language,

$L_{OI}(G) = \{t \in T_{\Sigma} \mid S \overset{*}{\underset{OI}{\Longrightarrow}} t\}$. By lemma 4.8.5, i

$S \overset{*}{\underset{OI}{\Longrightarrow}}_{G_1} t$, then $S \overset{*}{\underset{OI}{\Longrightarrow}}_{G_2} t$. Hence $L_{OI}(G_1) \subseteq L_{($

On the other hand, since $P_2 \subseteq P_1$, if $S \overset{*}{\underset{OI}{\Longrightarrow}}_{G_2}$ t

$S \overset{*}{\underset{OI}{\Longrightarrow}}_{G_1} t$ and hence $L_{OI}(G_2) = L_{OI}(G_1)$. Theref

$L_{OI}(G_1) = L_{OI}(G_2)$.

For convenience, the remainder of this th

assume that all tree grammars are weakly reduc

if it is not explicitly stated.

4.9 Weak Chomsky Normal Form

In a tree grammar, there is no "a priori"

the size of a right-hand side of a production.

can be simplified if these right hand sides ar

such that the number of terminal and nontermin

symbols occurring in the tree, are bounded by

two. This section presents one form of this t

tree grammar, as well as the method to transfo

tree grammar into this form.

A tree grammar $G=(\Phi,\Sigma,P,S)$ is in <u>weak Chomsk</u>
<u>normal form</u> if and only if for each production p∈
is in one of the following three forms:

i) $F(\vec{x})\text{->}t$ where for all $u\in(\mathrm{dom}(t)-\mathrm{var}(t))$, $t(u)\in\Phi$, and $l_{\Phi}(t)=2$;

ii) $F(\vec{x})\text{->}f(x'_1,\ldots,x'_q)$ where $f\in\Sigma$, $q=r(f)$, ar all i, $1\leq i\leq q$, $x'_i\in\mathbf{X}_{r(F)}$;

iii) $F(\vec{x})\text{->}x'$ where $x'\in\mathbf{X}_{r(F)}$

(**Note:** The notion of weak Chomsky normal form originates from the definition of Chomsky normal for context-free string grammars, see Chomsky[59]

In other words, the tree grammar G has the follow properties:

i) NT/T segmented,

ii) 2 - normal form, and

iii) derivation-renaming free.

Also, note that "erasing" in either form (via eps rules or nonconservative productions) still exist

__Example 4.9.1:__ Let $G_1=(\Phi_1,\overline{\Sigma},P_1,S_1)$ and $G_2=(\Phi_2,\overline{\Sigma},P_2,S_2)$ be tree grammars such that

$\Phi_1 = \{S_1,S_2,F,\hat{a},\hat{f}\}$ where $r(S_1)=0$, $r(S_2)=0$, $r(\hat{a})=0$, and $r(\hat{f})=2$;

$\overline{\Sigma} = \{a,f\}$ where $r(a)=0$ and $r(f)=2$;

```
P1  = { S2->S1 , S1->F , F->F ,
                    |    |  |    |
                    a    x  f^
                           / \
                          x   x

      F->x , f^ -> f ,  a^->a};
      |      / \  / \
      x     x  y  x  y
```

$\Phi_2 = \{S_2,F\}$ where $r(S_2)=0$ and $r(F)=1$; and

```
P2  = { S2->F , F->F , F->x}.
            |    |  |    |
            a    x  f    x
                   / \
                  x   x
```

Then, G_1 is in weak Chomsky normal form whi

not.

To show that one can convert any tree gra into a tree grammar G_2 such that G_2 is in weak normal form, we will use the above properties. However, first one must show that these proper sufficient in showing that a tree grammar is i Chomsky normal form.

mma 4.9.1: Given a tree grammar $G=(\Phi,\Sigma,P,S)$ such is

i) NT/T segmented,

ii) 2 - normal form, and

iii) derivation-renaming free,

en G is in weak Chomsky normal form.

oof: Assume G is not in weak Chomsky normal form. en there exists a production $F(\vec{x})\rightarrow t\in P$ such that ither

1) for all $v\in(dom(t)-var(t))$ $t(u)\in\Phi$ and $l_{\Phi}(t)$

2) $t=f(x_1',\ldots,x_q')$ where $q=r(f)$ and for all i, $1\leq i\leq q$, $x_i'\in X_{r(F)}$

3) $F(\vec{x})\rightarrow x'$ where $x'\in X_{r(F)}$

nce G is NT/T segmented either

a) for all $u\in(dom(t)-var(t))$ $t(u)\in\Phi$

b) $t(\varepsilon)\in\Sigma$ and $(dom(t)-var(t))=\{\varepsilon\}$.

nce, the only way that $F(\vec{x})\rightarrow t$ can not be in we omsky normal form is if for all $u\in(dom(t)-var(t))$ $u)\in\Phi$, $l_{\Phi}(t)>0$, and $l_{\Phi}(t)\neq 2$. Since G is in 2-nor

form, clearly $l_{\bar{\Phi}}(t)\leq 2$. Hence, it must be the
$l_{\bar{\Phi}}(t)=1$. Since G is derivation-renaming free,
$= \{t \mid t \in T_{\bar{\Phi}}(X_A)$ and $l_{\bar{\Phi}}(t)=1\}$, which is a contr
Hence, G is in weak Chomsky normal form.

The method to transform any tree grammar
weak Chomsky normal form is as follows:

i) Let G_2 be the NT/T segmented grammar

ii) Let $G_3,\ldots,G_q$ be tree grammars such t
in 2-normal form and for each G_i, $3\leq i$
$G_i=(\bar{\Phi}_i,\bar{\Sigma},P_i,S_i)$ where $G_i=\text{reduced}_n(G_{i-}$
for some $n>2$, $p_{i-1}\in P_{i-1}$, and p_{i-1} is
form $F(\vec{x})\text{->}t$ such that
$t\in \text{over}_{\bar{\Sigma}\vee\bar{\Phi}_{i-1},n-1}(\text{rhs}(P_{i-1}))$.

iii) $G_{q+1}=\text{drf}(G_q)$ where G_{q+1} is in weak Ch
normal form.

To show that these transformations are co
the following three lemmas and theorem are pre

<u>Lemma</u> <u>4.9.2:</u> Given any tree grammar $G_1=(\bar{\Phi}_1,\bar{\Sigma},P$
where G_1 is NT/T segmented and in n-normal for
production $F(\vec{x})\text{->}t\in P_1$ such that
$t\in \text{over}_{\bar{\Sigma}\vee\bar{\Phi}_1,n-1}(\text{rhs}(P_1))$, and $G_2=\text{reduced}_n(g_1,F$

:hen G_2 is also NT/T segmented.

<u>»roof;</u> Assume G_2 is not NT/T segmented. Then, th" exists a production G(l?)->s€P_2 such that neither

i) For all u€(dom(s)-var(s)),s(u)€L_2.

ii) s(e)€$\bar{i}$ and (dora(s)-var(s))={€}.

Jy the definition of G_2, P_2»(P_1-{F(lf)->t)}) V {S_2 ?(*)->S_{rhs}(G_1,F(1f)->t), T(it)->E_{NT}(G_1,F(jf)->t)} wh ONT(t). Since G_1 is NT/T segmented, clearly ;(lT)->s«5P_1. Clearly, by the definition of NT/T segmented, S_2->S_{1L} is NT/T segmented. Similarly, ?(^)->t€P_1 and G_1 is NT/T segmented, both ?(#)->S_{rhs}(G_1,F(lf)->t) and T(#)->E_{NT}(G_1,F(ie)->t) sTT/T segmented. But then G(l?)->s must be NT/T segmented which is a contradiction. Therefore G_2 tfT/Tsegmented.

<u>Lemma</u> <u>4.9.3:</u> Given any two tree grammars G_1 and G where G_1"(ij$\bar{i}$jP_1,S) is NT/T segmented and G_2=drf(G_1)^{as}(i,$\bar{i}$,P_2,S), then G_2 is NT/T segmented.

<u>Proof</u>: Assume G_2 is not NT/T segmented. Then the exists a production H(l?)~>t€P_2 such that neither

i) for all $u \in (dom(t)-var(t))$, $t(u) \in \overline{\Phi}$

ii) $t(e) \in \overline{\Sigma}$ and $(dom(t)-var(t))=\{\varepsilon\}$.

Since G_1 is NT/T segmented, it must be the case t
$H(\vec{x})->t \notin P_1$. Hence, by the definition of $drf(G_1)$,
must be the case that for some $t_n \in I_H$ such that H(
$t_1 \overline{\overline{OI}}> t_2 \overline{\overline{OI}}> \ldots \overline{\overline{OI}}> t_n$ and $t_n \overline{\overline{OI}}> t$ using $G(\vec{x})$
where $t_1,\ldots,t_n \in SN(\Phi)$ and $s \notin SN(\Phi)$. But then
$t_n=G(x_1',\ldots,x_q')$ where $q=r(G)$ and $x_1',\ldots,x_q' \in \mathbf{X}_A$.
Clearly, $(dom(s)-var(s))=(dom(t)-var(t))$. Depend
$G(\vec{x})->s$, there are two cases:

case 1: For all $u \in (dom(s)-var(s))$, $s(u) \in \overline{\Phi}$. Clea
for all $u \in (dom(t)-var(t))$, $t(u) \in \overline{\Phi}$ which is a
contradiction.

case 2: $s(e) \in \overline{\Sigma}$ and $(dom(s)-var(s))=\{e\}$. Clearly
$s(e)=t(e)$ which is a contradiction.

Hence G_2 is NT/T segmented.

Lemma 4.9.4: Given any two tree grammar G_1 and G_2
that $G_1=(\overline{\Phi},\overline{\Sigma},P_1,S)$ is in n-normal form and
$G_2=drf(G_1)=(\overline{\Phi},\overline{\Sigma},P_2,S)$, then G_2 is in n-normal for

Proof: Assume G_2 is not in n-normal form. Then t
exists a production $F(\vec{x})->t \in P_2$ such that $1_{\overline{\Sigma} \vee \overline{\Phi}}(t)>$
Since G_1 is in n-normal form, $F(\vec{x})->t \notin P_1$. Hence,
must be the case that for some $t_m \in I_F$ that $F(\vec{x}) \overline{\overline{OI}}$

$\overline{\overline{OI}}>$ t_2 $\overline{\overline{OI}}>$ ... $\overline{\overline{OI}}>$ t_m and t_m $\overline{\overline{OI}}>$ t using $G(\vec{x})\text{->}s$ where $t_1,t_2,\ldots,t_m \in SN(\Phi)$ and $s \notin SN(\Phi)$. But then $t_m=G(x'_1,\ldots,x'_q)$ and $t=s(x'_1,\ldots,s'_q)$ where $q=r(F)$ a $x'_1,\ldots,x'_q \in \mathbf{X}_A$. Clearly $l_{\overline{\Sigma V \Phi}}(t)=l_{\overline{\Sigma V \Phi}}(s)$. However since G_1 is in n-normal form and $G(\vec{x})\text{->}s \in P_1$, $l_{\overline{\Sigma V \Phi}}$ which is a contradiction. Therefore G_2 is in n-n form.

Theorem 4.9.1: Given a tree grammar $G=(\overline{\Phi},\overline{\Sigma},P,S)$, there exists an algorithm to generate a tree gram such that $L_{OI}(G) = L_{OI}(G')$ and G' is in weak Chom normal form.

Proof: By theorem 4.4.1, there exists a tree gram such that G_1 is the NT/T segmented grammar of G_1 $L_{OI}(G_1) = L_{OI}(G)$. By theorem 4.5.1, there exists finite sequence of tree grammars $G_1,\ldots,G_q$ such t

i) $G_i = (\overline{\Phi}_i,\overline{\Sigma},P_i,S_i)$ for all i, $1 \leq i \leq q$;

ii) for each i, $1<i\leq q$, there exists an $n>2$ s that $G_i=\text{reduced}_n(G_{i-1},p_{i-1})$ where $p_{i-1} \in P$ a production of the form $F(\vec{x})\text{->}t$ and $t \in \text{over}_{\overline{\Sigma V \Phi}_{i-1},n-1}(\text{rhs}(P_{i-1}))$;

iii) for all i, $1<i\leq q$, $L_{OI}(G_i) = L_{OI}(G_{i-1})$;

iv) G_q is in 2-normal form.

By lemma 4.9.2, for all i, $1<i\leq q$, G_i is also NT/
segmented. Let G'=drf(G_q). By theorem 4.6.1,
$L_{OI}(G') = L_{OI}(G_q)$. By lemmas 4.9.3 and 4.9.4, G
also NT/T segmented and 2-normal. Since G' is N
segmented, 2-normal, and derivation-renaming fre
lemma 4.9.1 states that G' is in weak Chomsky no
form.

4.10 Leaf-linear Tree Grammars

This section presents a restricted form of
grammars known as leaf-linear tree grammars. Th
grammars are called leaf-linear because nontermi
can only occur as leaves on the right-hand side
productions in the tree grammar. In other words
tree grammar is <u>leaf-linear</u> if and only if the r
every nonterminal is zero. Furthermore, since t
of all nonterminals is zero, there are no variab
leaf-linear tree grammars.

The main reason that leaf-linear tree gramma of interest is that the class of tree languages generated by leaf-linear tree grammars is identic the class of regular tree languages (see Brainerd or Doner[70]). One should note that this result leaf-linear tree grammars corresponds to the resu about left-linear context-free string grammars (i the class of left-linear string grammars is ident to the class of regular string languages, see Bar-Hillel and Shamir[60]).

It is a well known fact that the class of re tree languages is strictly contained within the c of context-free tree languages (see Rounds[70]) T result can easily be shown by the use of a pumpin lemma for regular trees based on the pumping lemm presented by Rabin and Scott[59] (see Thatcher[73 The following theorem presents (without proof) a of the pumping lemma based on the assumption that tree language is generated by a tree grammar.

Theorem 4.10.1: Given any tree grammar $G=(\Phi,\overline{\Sigma},P,S$ $L(G)$ is regular, then for any tree $t \in L(G)$ such th depth(t)>sum{depth(s) | $F(\vec{x}) \rightarrow s \in P$}, there exists $t_1, t_2 \in T_{\overline{\Sigma}}(X_1)$ and $t_3 \in T_{\overline{\Sigma}}$ such that $|var(t_1)|=|var(t$ depth(t_2)>0, $t=t_1(t_2(t3))$, and for all $n \geq 0$,

$t_1[(t_2)(t_3)]^n \in L(G)$.

Example 4.10.1: Let $G=(\Phi,\Sigma,P,S)$ be a tree gram

that

Φ = {S,F} where r(S)=0 and r(F)=1;

Σ = {a,f,g} where r(a)=0, r(f)=1, and r(g)=

```
P = {S->F,  F->F,  F->  g  }
        |   |   |   |   / \
        a   x   f   x x    x
                |
                x
```

Then, L(G) is not a regular tree language. Th

the definition of a tree language, all trees i

are in the form

```
           g
         /   \
      f         f
      |         |
      .         .
  n   .         .   n        where n≥0
      .         .
      |         |
      f         f
      |         |
      a         a
```

Assume L(G) is a regular tree language. Hence

theorem 4.10.1, for any $k \geq 4$ such that $t \in L(G)$ a

depth(t)=k, the theorem must apply. Consideri

possible values for t_1,t_2 and t_3, there are 4

case 1:

let $t_1=x$, $t_2 = \begin{array}{cc} \multicolumn{2}{c}{g,} \\ / & \backslash \\ n_1\left\{\begin{array}{c}f\\|\\ \vdots \\|\\f\end{array}\right. & \left.\begin{array}{c}f\\|\\ \vdots \\|\\f\end{array}\right\}k-1 \\ | & | \\ x & a \end{array}$ $t_3 = \begin{array}{c} \left.\begin{array}{c}f\\|\\ \vdots \\|\\f\end{array}\right\}n_2 \\ | \\ a \end{array}$ where $n_2>0$ and n_1+n

But then $t_1[(t_2)(t_3)]^0 = \begin{array}{cc} \multicolumn{2}{c}{g} \\ / & \backslash \\ n_1<k-1\left\{\begin{array}{c}f\\|\\ \vdots \\|\\f\end{array}\right. & \left.\begin{array}{c}f\\|\\ \vdots \\|\\f\end{array}\right\}k-1 \\ | & | \\ a & a \end{array}$

which is not in L(G).

case 2:

let $t_1=x$, $t_2 = \begin{array}{cc} \multicolumn{2}{c}{g,} \\ / & \backslash \\ k-1\left\{\begin{array}{c}f\\|\\ \vdots \\|\\f\end{array}\right. & \left.\begin{array}{c}f\\|\\ \vdots \\|\\f\end{array}\right\}n_1 \\ | & | \\ a & x \end{array}$ $t_3 = \begin{array}{c} \left.\begin{array}{c}f\\|\\ \vdots \\|\\f\end{array}\right\}n_2 \\ | \\ a \end{array}$ where $n_2>0$ and n_1+n

But then $$t_1[(t_2)(t_3)]^0 = \begin{array}{ccc} & g & \\ / & & \backslash \\ k-1\left\{\begin{array}{c} f\\ |\\ \cdot\\ \cdot\\ \cdot\\ |\\ f\end{array}\right. & & \left.\begin{array}{c} f\\ |\\ \cdot\\ \cdot\\ \cdot\\ |\\ f\end{array}\right\} n_1<k-1 \\ | & & | \\ a & & a \end{array}$$

which is not in L(G).

<u>case</u> <u>3:</u>

Let $$t_1 = \begin{array}{ccc} & g & \\ / & & \backslash \\ n_1\left\{\begin{array}{c} f\\ |\\ \cdot\\ \cdot\\ \cdot\\ |\\ f\end{array}\right. & & \left.\begin{array}{c} f\\ |\\ \cdot\\ \cdot\\ \cdot\\ |\\ f\end{array}\right\} k-1 \\ | & & | \\ x & & a \end{array}, \quad t_2 = \begin{array}{c} \left.\begin{array}{c} f\\ |\\ \cdot\\ \cdot\\ \cdot\\ |\\ f\end{array}\right\} n_2 \\ | \\ x \end{array}, \quad t_3 = \begin{array}{c} \left.\begin{array}{c} f\\ |\\ \cdot\\ \cdot\\ \cdot\\ |\\ f\end{array}\right\} n_3 \\ | \\ a \end{array}$$

where $n_2>0$ and $n_1+n_2+n_3=k-1$.

But then $$t_1[(t_2)(t_3)]^0 = \begin{array}{ccc} & g & \\ / & & \backslash \\ n_1+n_3<k-1\left\{\begin{array}{c} f\\ |\\ \cdot\\ \cdot\\ \cdot\\ |\\ f\end{array}\right. & & \left.\begin{array}{c} f\\ |\\ \cdot\\ \cdot\\ \cdot\\ |\\ f\end{array}\right\} k-1 \\ | & & | \\ a & & a \end{array}$$

which is not in L(G).

e 4:

```
 t1 = g ,    t2 = f-,  t3 = f-
     / \            |  |        |  |
   -f   f-          .  |        .  |
  | |   | |         .  } n2     .  } n3
  | .   . |         .  |        .  |
k-1{ .  .  }n1      |  |        |  |
  | .   . |         f-          f-
  | |   | |         |           |
   -f   f-          x           a
    |   |
    a   x
```

re $n_2>0$ and $n_1+n_2+n_3=k-1$.

then $t_1[(t_2)(t_3)]^0$ =

```
                 g
               /   \
             -f     f-
            | |     | |
            | .     . |
       k-1 {  .     .  } n1+n3<k-1
            | .     . |
            | |     | |
             -f     f-
              |     |
              a     a
```

ch is not in L(G).

ce, by theorem 4.10.1, L(G) is not a regular tree guage.

orem 4.10.2: The class of regular tree languages roper subset of the context-free tree languages.

of: Since the class of regular tree languages is ntical to the class of tree languages generated b f-linear tree grammars, there exists a leaf-linea e grammar to generate any regular tree language.

Hence, since every leaf-linear tree grammar is
grammar, the class of regular tree languages is
contained in the class of context-free tree lan
From example 4.10.1, clearly there exists a context-free tree language which is not regular. Therefore, the class of regular tree languages is a proper subset of the class of context-free tree languages.

4.11 **Root-linear Tree Grammars**

This section presents a restricted form of grammars known as root-linear tree grammars. Root-linear tree grammars are called root-linear because nonterminals occurring on the right-hand of productions can only occur at the root. A production $F(\vec{x}) \to t$ is <u>root-linear</u> if and only if $l_\Phi(t)=0$, or $l_\Phi(t)=1$ and $t(\varepsilon)\in\Phi$. Similarly, a tree grammar $G=(\Phi,\Sigma,P,S)$ is <u>root-linear</u> if and only if every production $p\in P$, p is root-linear.

A well known result is that the class of string languages generated by right-linear string grammars is identical to the class of regular string languages which is identical to the class of string languages generated by left-linear string grammars. Hence

ght assume that the same results apply to leaf-li
d root-linear tree grammars and hence, both gener
gular tree languages. However, this is not the c

The tree language generated by a root-linear t
ammar is called a <u>coregular tree language</u> and the
mainder of this section presents several results
out that the class of coregular tree languages (s
nold and Dauchet[76]). Theorem 4.11.1 shows that
ass of coregular tree languages is not contained
e class of regular tree languages. Theorem 4.11.
esents a pumping lemma for coregular tree languag
ich can be used to test if a tree language is not
regular tree language. The section concludes wit
eorem 4.11.3 which uses the pumping lemma to show
at the class of coregular tree languages is a pro
bset of the class of context-free tree languages.

<u>eorem 4.11.1:</u> The class of coregular tree languag
not contained in the class of regular tree
nguages.

<u>oof:</u> Example 4.10.1 presents a root-linear tree
ammar and shows that the language generated by th
ot-linear tree grammar is not a regular tree
nguage. Hence, the class of coregular tree langu

cannot be in the class of regular tree languages.

The next lemma and theorem present a new res which is believed to be the first known form of a pumping lemma for the class of coregular tree lan (one should note that Arnold and Dauchet[76] also present a pumping lemma for coregular tree langua However, thier result is on language duplication unrelated to the lemma presented here).

Lemma 4.11.1: Given a root-linear tree grammar $G=(\Phi,\Sigma,P,S)$, any nonterminal $H\in\Phi$ where $r(H)=m$, an sequence of trees $s_1,\ldots,s_m \in T_\Sigma$, if $H(s_1,\ldots,s_m) =$ $F(t_1,\ldots,t_q)$ where $r(F)=q$, then there exists a se of trees $t'_1,\ldots,t'_q \in T_\Sigma(X_q)$ such that $t_i=t'_i(s_1,\ldots,$ for all i, $1\leq i\leq q$.

Proof: By induction on n.

base case: $n=0$. Trivial. Let $(t'_1,\ldots,t'_q)=(x_1,\ldots,x_q)$.

Inductive step: $F(p_1,\ldots,p_m) \Longrightarrow^n H(s_1,\ldots,s_q) =$ $K(t_1,\ldots,t_k)$ where $r(F)=m$, $r(H)=q$, and $r(K)=k$. B induction, there exists a sequence of trees $t''_1,\ldots,t''_q \in T_\Sigma(X_q)$ such that $s_i=t''_i(p_1,\ldots,p_m)$ and s or all i, $1 \leq i \leq q$. By definition of the last deriv tep, there exists a production of the form $H(\vec{x})->$ here $\beta(\mathcal{E})=K$ and $H(s_1,\ldots,s_q) \Longrightarrow \beta(s_1,\ldots,s_m) =$ $(t_1,\ldots,t_k)$. For all i, $1 \leq i \leq k$, let $t_i=\beta/i(s_1,\ldots$ ince G is root-linear, clearly, $t_i \in T_{\Sigma}$ for all i, $\leq i \leq k$. By substituting the values for s_1 through $_i=\beta/i(t''_1(p_1,\ldots,p_m), \ldots, t''_q(p_1,\ldots,p_m))$ for all $\leq i \leq k$. Hence, by lemma 2.8.2, for all i, $1 \leq i \leq k$, $_i=\beta/i(t''_1,\ldots,t''_q)(p_1,\ldots,p_m)$.

heorem 4.11.2 - The pumping lemma for coregular t anguages: Given any root-linear tree grammar $=(\Phi,\Sigma,P,S)$; any $t \in L(G)$ such that epth(t)>sum{depth(s) | $F(\vec{x})->s \in P$}; there exists

i) some nonterminal $F \in \Phi$ where $r(F)=m$,

ii) a tree $t' \in T_{\Sigma}(X_m)$,

iii) a sequence of trees $t_1,\ldots,t_m \in T_{\Sigma}(X_m)$, an

iv) a sequence of trees $s_1,\ldots,s_m \in T_{\Sigma}$

uch that

a) $t=t'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^1$

b) for all $n\geq 0$ $t'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]$

c) $t'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^0 \neq$
$t'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^1$

d) for all $n\geq 0$ $S \Longrightarrow^* F[(t_1,\ldots,t_m)(s_1,\ldots$

Proof: Since depth(t)>sum{depth(s) | $F(\vec{x})\to s\in P$} must be the case that some production is used m once, and the production will increase the dept tree generated. That is, it must be the case t $S \Longrightarrow^{n_1} F(s_1,\ldots,s_m) \Longrightarrow^{n_2} F(s_1',\ldots,s_m') \Longrightarrow^{n_3}$ $t'(s_1',\ldots,s_m') = t$ where $F\in\Phi$, for all i, $1\leq i\leq 3$, and there exists a k, $1\leq k\leq m$, such that $depth(s_k)<depth(s_k')$ and for some $u\in var(t')$, $t'($ By lemma 4.11.1, there exist trees $t_1,\ldots,t_m$ su $s_i'=t_i(s_1,\ldots,s_m)$ for all i, $1\leq i\leq m$. By definiti $t'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^0 = t'(s_1,\ldots,s_m)$ and $t'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^1 = t'(t_1(s_1,\ldots,s_m),$ $t_m(s_1,\ldots,s_m))$. But then $t'(t_1(s_1,\ldots,s_m), \ldots,$ $t_m(s_1,\ldots,s_m)) = t'(s_1',\ldots,s_m')$. Clearly $t'(s_1,$ $\neq t'(s_1',\ldots,s_m')$ since $t'(s_1',\ldots,s_m')/u=s_k'$, $t'(s_1,\ldots,s_m)/u=s_k$, and $depth(s_k)<depth(s_k')$. R the above information, one can conclude that

1) $S \Longrightarrow^* F(s_1,\ldots,s_m)$

2) $F(\vec{x}) \Longrightarrow^+ F(t_1,\ldots,t_m)$

3) $F(\vec{x}) \Longrightarrow^* t'(x_1,\ldots,x_m)$

4) $t'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^0 \neq$
$t'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^1$

5) $t = t'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^1$ where
$s_1,\ldots,s_m \in T_\Sigma$ and $t',t_1,\ldots,t_m \in T_\Sigma(X_m)$.

o show the remaining portions of the theorem, let
nductive hypothesis be that for any $n \geq 0$,
$'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^n \in L(G)$ and $S \Longrightarrow^*$
$[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^n$. Then, by using proof
nduction, the following cases exist:

<u>ase case:</u> $n=0$. By condition (1), $S \Longrightarrow^*$
$(s_1,\ldots,s_m)$. By condition (3) $F(s_1,\ldots,s_m) \Longrightarrow^*$
$'(s_1,\ldots,s_m)$. Hence $t'(s_1,\ldots,s_m) \in L(G)$. By the
efinition of the n^{th} m-way composition, $F(s_1,\ldots,$
$[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^0$ and $t'(s_1,\ldots,s_m) =$
$'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^0$.

<u>nductive step:</u> $n>0$. By induction, $S \Longrightarrow^*$
$[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^n$. By condition (2),
$[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^n \Longrightarrow^+$
$(t_1,\ldots,t_m)[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^n$. By lemma 2

$F(t_1,\ldots,t_m)[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^n =$
$F(t_1[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^n, \ldots,$
$t_m[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^n)$. By inspection o
definition of the $n+1^{th}$ m-way composition, cle
$F(t_1[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^n, \ldots,$
$t_m[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^n) =$
$F[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^{n+1}$. By condition (3
$F[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^{n+1} \Longrightarrow^+$
$t'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^{n+1}$. By inspection
previous construction, clearly
$t'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^{n+1} \in T_{\overline{\Sigma}}$. Hence, by t
definition of the tree language generated by G
$t'[(t_1,\ldots,t_m)(s_1,\ldots,s_m)]^{n+1} \in L(G)$.

Theorem 4.11.3: The class of coregular tree la
is a proper subset of the class of context-fre
languages.

Proof: Since every root-linear tree grammar is
grammar, clearly the class of coregular tree l
is contained in the class of context-free tree
languages. To show that there exists a tree l
which is not a coregular tree language, consid
tree grammar $G=(\overline{\Phi},\overline{\Sigma},P,S)$ such that

$\overline{\Phi} = \{S,F\}$ where $r(S)=0$ and $r(A)=1$;

$\overline{\Sigma} = \{a,f,g\}$ where $r(a)=0$ and $r(f)=r(g)=1$;

```
P = {S->A,  A->x,  A->f}.
        |   |      |   |
        a   x      x   A
                       |
                       g
                       |
                       x
```

tree language generated by G is the set of trees

form

```
f
|
.
.     n
.
|
f
|            where n≥0.
g
|
.
.     n
.
|
g
|
a
```

ume $L(G)$ is a coregular tree language. Then by eorem 4.11.2, for any $t \in L(G)$ where $depth(t)>5$, the st exist trees $t_1, t_2, t_3 \in T_{\Sigma}(X_1)$ where $depth(t_2)>0$, $_1[(t_2)(t_3)]^1$, $t_1[(t_2)(t_3)]^0 \neq t_1[(t_2)(t_3)]^1$, and . $n \geq 0$, $t_1[(t_2)(t_3)]^n \in L(G)$. Let $t \in L(G)$ where th$(t)=2k+1$ for any $k>2$. Then, there are 5 cases sider in choosing trees t_1, t_2, and t_3.

case 1:

$$t_1 = \begin{array}{c} \left.\begin{array}{c} f \\ | \\ \cdot \\ \cdot \\ \cdot \\ f \end{array}\right\} n_1 \\ | \\ x \end{array} \qquad t_2 = \begin{array}{c} \left.\begin{array}{c} f \\ | \\ \cdot \\ \cdot \\ \cdot \\ f \end{array}\right\} n_2 \\ | \\ x \end{array} \qquad t_3 = \begin{array}{c} \left.\begin{array}{c} f \\ | \\ \cdot \\ \cdot \\ \cdot \\ f \end{array}\right\} n_3 \\ | \\ \left.\begin{array}{c} g \\ | \\ \cdot \\ \cdot \\ \cdot \\ | \\ g \end{array}\right\} k \\ | \\ a \end{array}$$

where $n_1+n_2+n_3=k$ and $n_2>0$. But then $t_1[(t_2)($
have $n_1+n_3<k$ nodes labeled by f followed by k
labeled by g. Hence, $t_1[(t_2)(t_3)]^0 \notin L(G)$.

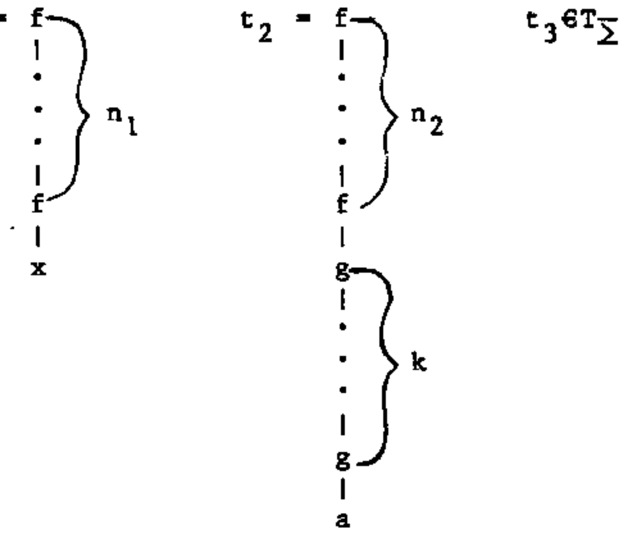
case 2:

$$t_1 = \begin{array}{c} \left.\begin{array}{c} f \\ | \\ \cdot \\ \cdot \\ \cdot \\ | \\ f \end{array}\right\} n_1 \\ | \\ x \end{array} \qquad t_2 = \begin{array}{c} \left.\begin{array}{c} f \\ | \\ \cdot \\ \cdot \\ \cdot \\ | \\ f \end{array}\right\} n_2 \\ | \\ \left.\begin{array}{c} g \\ | \\ \cdot \\ \cdot \\ \cdot \\ | \\ g \end{array}\right\} n_3 \\ | \\ x \end{array} \qquad t_3 = \begin{array}{c} \left.\begin{array}{c} g \\ | \\ \cdot \\ \cdot \\ \cdot \\ | \\ g \end{array}\right\} n_4 \\ | \\ a \end{array}$$

where $n_1+n_2=n_3+n_4=k$ and $n_2+n_3>0$. But then in

$[(t_2)(t_3)]^2$, there will exist n_1+n_2 nodes labeled. 1
followed by n_3 nodes labeled by g, followed by n_2
is labeled by f, followed by n_3+n_4 nodes labeled 1
Clearly, the only way that $t_1[(t_2)(t_3)]^2 \in L(G)$ is
ler $n_2=0$ or $n_3=0$. Assume $n_2=0$. Then, there woul
: nodes labeled by f followed by $n_3+n_3+n_4>k$ nodes
sled by g. Hence $n_2 \neq 0$. By a similar argument,
). Therefore $t_1[(t_2)(t_3)]^2 \notin L(G)$.

e 3:



re $n_1+n_2=k$ and $n_2>0$. However, $t_1[(t_2)(t_3)]^0 =$
$[(t_2)(t_3)]^1$ and hence this case does not apply.

case 4:

$$t_1 = \begin{array}{c} \left.\begin{array}{c} f\\ |\\ \cdot\\ \cdot\\ \cdot\\ f\end{array}\right\} k \\ | \\ \left.\begin{array}{c} g\\ |\\ \cdot\\ \cdot\\ \cdot\\ |\\ g\end{array}\right\} n_1 \\ | \\ x \end{array} \qquad t_2 = \begin{array}{c} \left.\begin{array}{c} g\\ |\\ \cdot\\ \cdot\\ \cdot\\ g\end{array}\right\} n_2 \\ | \\ x \end{array} \qquad t_3 = \begin{array}{c} \left.\begin{array}{c} g\\ |\\ \cdot\\ \cdot\\ \cdot\\ g\end{array}\right\} n_3 \\ | \\ a \end{array}$$

where $n_1+n_2+n_3=k$ and $n_2>0$. But then $t_1[(t_2)($
have k nodes labeled by f, followed by $n_1+n_3<$
labeled by g. Hence $t_1[(t_2)(t_3)]^0 \notin L(G)$.

case 5:

$$t_1 = \begin{array}{c} \left.\begin{array}{c} f\\ |\\ \cdot\\ \cdot\\ \cdot\\ f\end{array}\right\} k \\ | \\ \left.\begin{array}{c} g\\ |\\ \cdot\\ \cdot\\ \cdot\\ |\\ g\end{array}\right\} n_1 \\ | \\ x \end{array} \qquad t_2 = \begin{array}{c} \left.\begin{array}{c} g\\ |\\ \cdot\\ \cdot\\ \cdot\\ g\end{array}\right\} n_2 \\ | \\ a \end{array} \qquad t_3 \in T_{\Sigma}$$

where $n_1+n_2=k$ and $n_2>0$. However, $t_1[(t_2)(t_3)$
$t_1[(t_2)(t_3)]^1$ and hence this case cannot appl

fore, by theorem 4.11.2, G must not be a coregula
language. Furthermore, since G is a tree gramma
ly the class of coregular tree languages is a
r subset of the class of context-free tree
ages.

Chapter V

TREE PUSHDOWN AUTOMATA

This chapter presents a new model of a tre pushdown automaton, the (nondeterministic) bott tree automaton with tree pushdown stores (a TP short). The TPDA operates like a standard bott tree automaton, except that there is an interna which consists of a finite sequence of tree pus stores (or simply tree stacks). TPDAs correspo the standard (string) pushdown automata, in the manner that bottom-up tree automata correspond (string) finite automata. In other words, each stack is treated in the same manner as the stac pushdown automaton, in that a TPDA can only rea

oot of each tree stack, and nodes can be added
ed) or deleted (popped) at the roots of the tree
.

Section one begins by presenting a TPDA using
ard conventions. Section two presents a
ified form of the TPDA where the current state,
iated with each read-head, always labels the root
e corresponding tree stack. Hence, no explicit
ence of the states is needed and a "stateless"
pushdown automaton (a STPDA for short) is
duced. The chapter concludes by showing that the
of tree languages generated by tree grammars
an OI derivation, the class of tree languages
ted by TPDAs, and the class of tree languages
ted by STPDAs, are identical.

Tree Pushdown Automata

This section provides the definition of the tree
own automaton (the TPDA). A tree pushdown
aton is a bottom-up tree automaton augmented with
stacks, and the basic organization of a TPDA is
in figure 5.1.1.

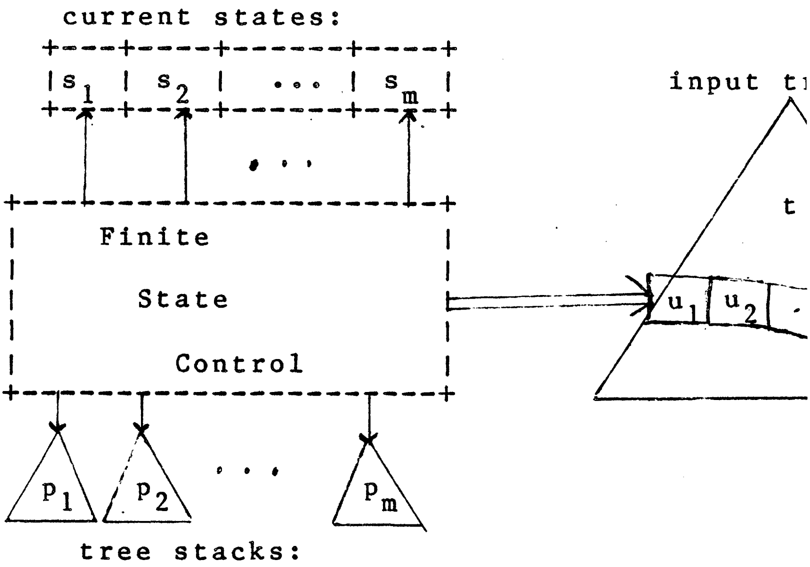


Figure 5.1.1 : Tree pushdown automata

Note: For each read-head u_i, there is exact corresponding current-state s_i, and one tree p_i.

Before describing a TPDA however, conside following informal description of a bottom-up automaton (for a more formal description of a tree automaton, see Buchi and Wright[60], Eile Wright[67], Doner[70], Thatcher and Wright[68] and Moran[69], Brainerd[69], and Thatcher[73]) bottom-up tree automaton $A=(S,\delta,s_0,Q)$ consists of states S, a transition map δ : tuples(S) -> initial state s_0, and a set of final states Q. this discussion, consider the input tree t (to

e) shown in figure 5.1.2.

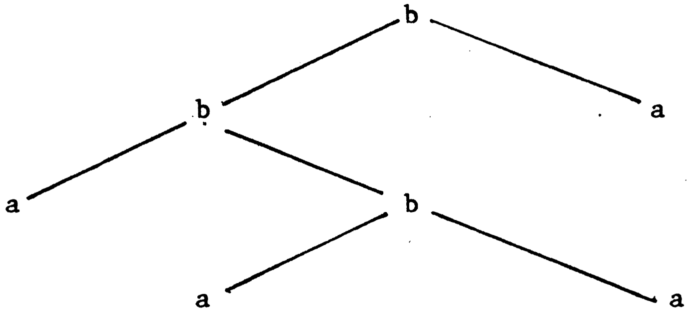


Figure 5.1.2

Sample input tree t using ranked alphabet Σ = {a,
where r(a)=0 and r(b)=2

A bottom-up tree automaton scans an input tree m its leaves, up to the root, verifying that the ut tree matches the pattern of the tree language. other words, the leaves of the input tree t are sidered as the starting points when scanning the ut tree t. Hence, below each leaf, a read-head (ker) is located, and the current state associated h each read-head is the initial state s_0. The tial configuration of the bottom-up tree automato graphically depicted in figure 5.1.3.

Reading (or scanning) the node immediatel read-head (i.e. a leaf) will cause the corres read-head to be advanced up to cover that node instance, the results of using the transition $s_1 \in \delta(s_0, a)$, applied (sequentially) to each of leftmost leaves in t, is shown in figure 5.1.4a - 5.1.4c. Note that the three leftmost read-heads are now covering the corresponding and that the states have been updated to s_1.

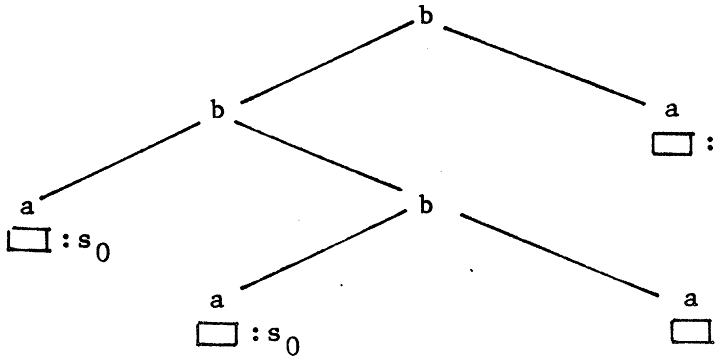


Figure 5.1.3

Initial configuration of a bottom-up tree a where ▭ denotes a read-head.

Reading an internal node requires that re cover each of its immediate descendants, and t corresponding states match the definition of t transition map δ. For instance, if $s_2 \in \delta((s_1, s$ $s_3 \in \delta((s_1, s_2), b)$, then figures 5.1.5a - 5.1.5b

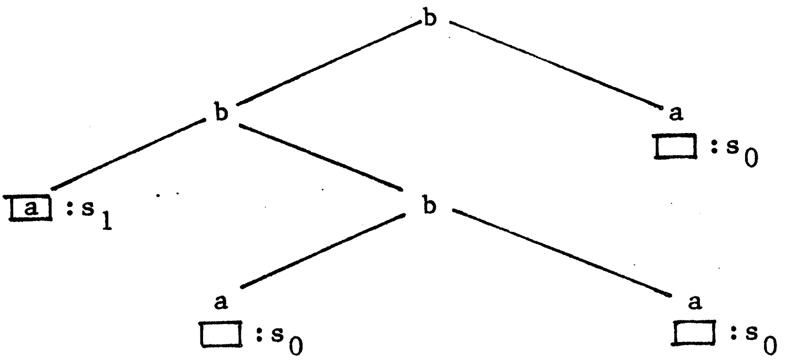


Figure 5.1.4a

Configuration of bottom-up automaton after readi
node 11 using transition $s_1 \in \delta(s_0,a)$

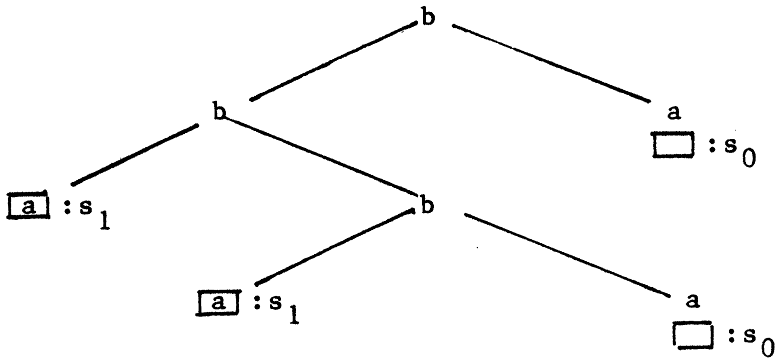


Figure 5.1.4b

Configuration of bottom-up automaton after readi
node 121 using transition $s_1 \in \delta(s_0,a)$

orresponding updates of the bottom-up tree automa
ote that in the process of advancing the read-hea
ver the nodes labeled with a "b", the read-heads

cated at each of its immediate descendants are m

ito one.

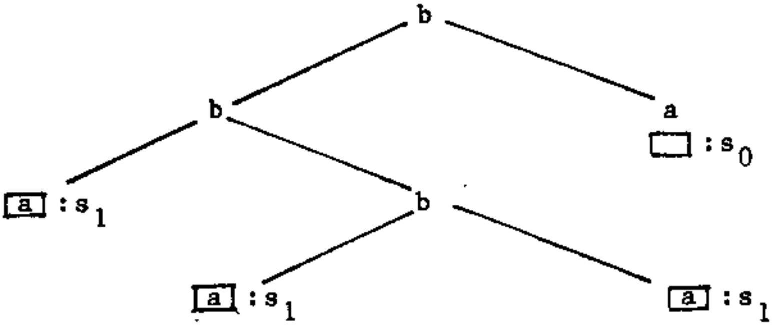


Figure 5.1.4c

Configuration of bottom-up automaton after readi
node 122 using transition $s_1 \in \delta(s_0,a)$

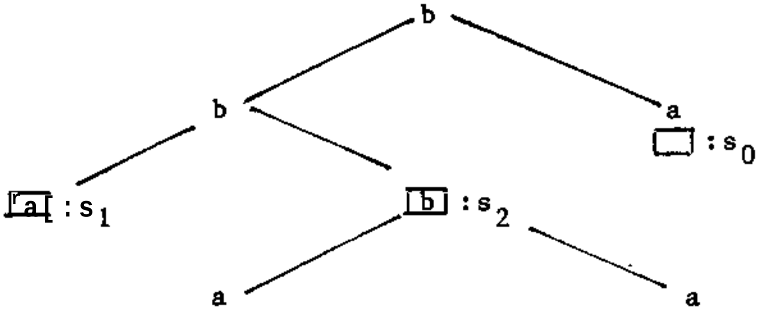


Figure 5.1.5a

Configuration of bottom-up automaton after readi
node 12 using transition $s_2 \in \delta((s_1, s_1),b)$

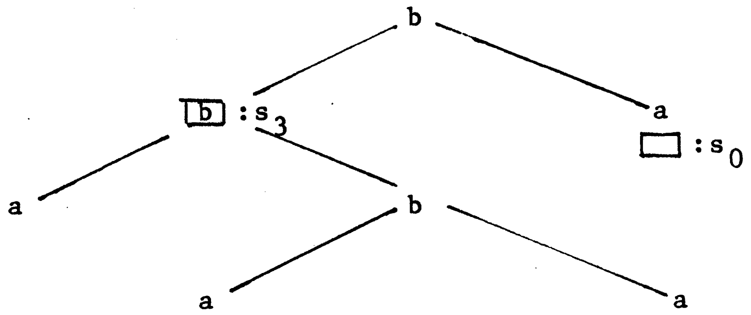


Figure 5.1.5b

Configuration of bottom-up automaton after readin
node 1 using transition $s_3 \in \delta((s_1,s_2),b)$

The process of advancing and merging read-head
cording to the transition map δ, continues until
ther there is no more legal moves, or the accepti
ndition is met. An accepting condition occurs
enever the read-heads have been advanced to the r
.e. there is only one read-head which covers the
ot) and its corresponding state is in the set of
nal states Q.

In a tree pushdown automaton, the bottom-up tr
tomaton is augmented with internal memory. The f
the internal memory is a sequence of trees (each
ee is called a tree stack), and each read-head ha
actly one tree stack. Each <u>tree stack</u> is a tree
fined by a ranked alphabet Γ called the <u>stack</u>

alphabet. The stack alphabet does not have to be distinct from the input alphabet Σ. Furthermore, a reserved constant $\perp$ denoting the empty tree stack is contained in Γ.

Intuitively, each tree stack is used to record the tree structure its corresponding read-head has scanned. In other words, when a TPDA is constructed to accept a tree language generated by a tree grammar G, the idea is to maintain each tree stack such that the tree stack is a subtree of some legal sentential form. Furthermore, whenever the tree stack matches the right-hand side of some production p in the tree grammar G, the inverse operation of a derivation step will be performed on the tree stack using the production p. Therefore, the general idea of the constructed TPDA is to have the read-heads advance toward the root of the input tree, transform the tree stacks using the inverse operation of a derivation step, until there is a single read-head at the root of the input tree and its corresponding tree stack is a one-node tree labeled by the start symbol of the tree grammar G.

Having provided insight into the notion of a
tack in a TPDA, a formal definition of a TPDA can
iven.

efinition 5.1.1: A TPDA is a 7-tuple
= $(S,\Sigma,\Gamma,\delta,s_0,\perp,Q)$ where

S is a finite set of states;

Σ is a finite ranked alphabet of input symbols;

Γ is finite ranked alphabet of stack symbols;

$\delta : (\{s_0\} \times \Sigma)$

$\vee$ (tuples(S) $\times \Sigma$)

$\vee$ ((S $\times T_{\Gamma}(X_m)$ $\times$ tuples(Γ)) $\times \{\varepsilon\}) \to 2^S$

is a function called the transition map wher
$m=\max\{r(\beta) \mid \beta \in \Gamma\}$, and δ has finite domain;

$s_0 \in S$ is the initial state;

$\perp \in \Gamma$ is a reserved constant denoting the
empty tree stack; and

$Q \subseteq S$ is the set of final states.

The transition map δ is defined such that all
efinitions are one of the following two forms:

__Shift-moves__ (read-moves):

i) $(q,F)\in\delta(s_0,a)$ where $q\in S$, $F\in\Gamma$, $a\in\bar{\Sigma}$, a

$r(a)=r(F)=0$

ii) $(q,F)\in\delta((q_1,\ldots,q_m),f)$ where $f\in\bar{\Sigma}$, $F\in$

$m=r(f)=r(F)>0$, and $q,q_1,q_2,\ldots,q_m\in S$

__Reduce-move__ (tree stack update only):

$(q_2,F)\in\delta((q_1,t,(F_1,\ldots,F_m)),\varepsilon)$ where $F\in\Gamma$,

$t\in T_{\Gamma}(X_m)$, $F_1,\ldots,F_m\in\Gamma$, and $q_1,q_2\in S$.

To emphasize that $(F_1,\ldots,F_m)$ is a look-back r

in the tree stack t, $\delta((q_1,t,(F_1,\ldots,F_m)),\varepsilon)$ w

denoted as $\delta((q_1,t\,[[F_1,\ldots,F_m\,]]),\varepsilon)$.

An __instantaneous__ __description__ (ID for shor

provides a "snapshot" description of a TPDA be

moves. An ID consists of a pair

$(a,t)\in 2^{S \times N^* \times T_{\Gamma}} \times T_{\bar{\Sigma}}$ where t is the input

a is a set of triples of the form (q,u,p) wher

the tree address of a node covered by a read-h

its corresponding state, and p is the tree sta

associated with that read head. Also, if u is

with a zero, it denotes the position immediate

the leaf at tree address u (i.e. corresponds

starting position of a read-head). The initia

configuration of a TPDA is the ID

$(s_0,u0,\perp) \mid u\in leaf(t)\},t)$ where t is the input tr
d the accepting configuration is an ID of the for
$(q,\varepsilon,\perp)\},t)$ where $q\in Q$ is a final state.

For example, let $D=(S,\Sigma,\Gamma,\delta,0,\perp,Q)$ be a TPDA
ere:

$S = \{0,1,2,3\}$;

$\Sigma = \{a,b\}$ where $r(a)=0$ and $r(b)=2$;

$\Gamma = \{F,a,b\}$ where $r(a)=0$ and $r(b)=r(F)=2$; and

$Q = \{2,3\}$.

If the input tree t is the tree shown in figu
1.1, then the initial ID is the instantaneous
scription

$_1 = (\{(0,110,\perp),(0,1210,\perp),(0,1220,\perp),(0,2,\perp)\},t$
igure 5.1.6 shows a graphical representation of
itial ID id_1). The two possible accepting
stantaneous descriptions are $id_2 = (\{(2,\varepsilon,\perp)\},t)$
$_3 = (\{(3,\varepsilon,\perp),t)$ which are graphically depicted
gures 5.1.7a and 5.1.7b. Finally, an example of
stantaneous description which is neither an initi
accepting ID is

$_4 = (\{(1,11,a),(2,12,F(a,b(a,a))),(0,2,\perp)\},t)$ wh
depicted in figure 5.1.8.

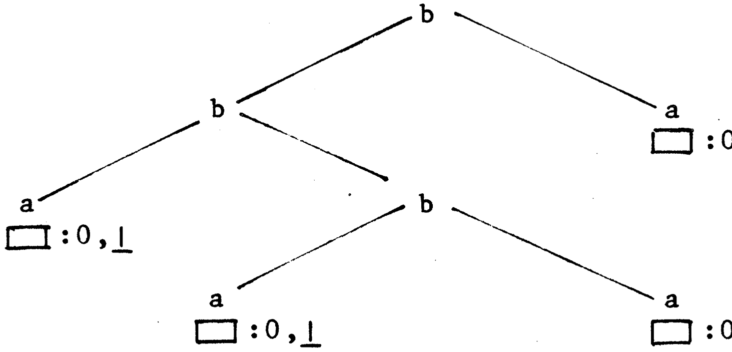


Figure 5.1.6

Graphical representation of an initial insta description where ▭ denotes the position o read-head and ":0,⊥" represents the correspo state and tree state, respectively, associat the corresponding read-head.

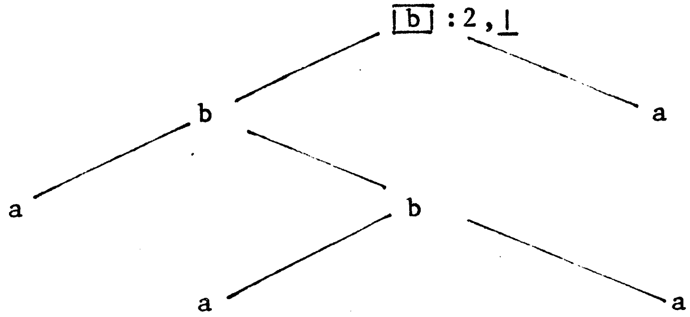
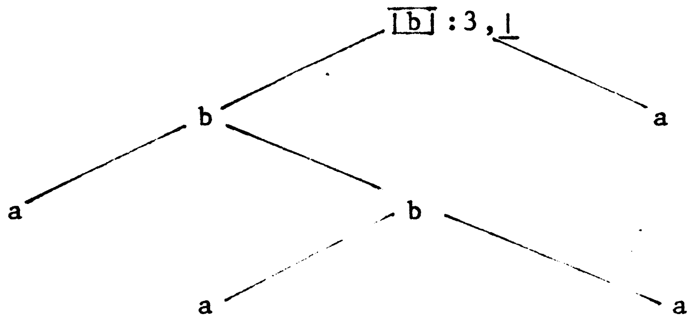


Figure 5.1.7a: Possible accepting instantane description



igure 5.1.7b: Possible accepting instantaneous
escription

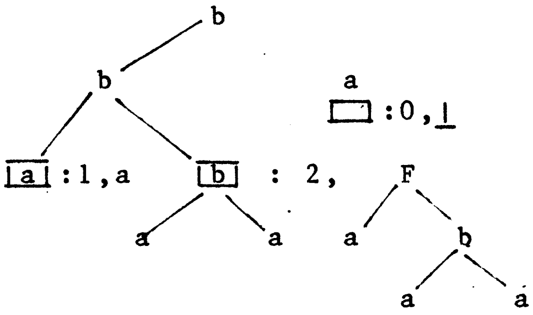


Figure 5.1.8

ample legal instantaneous description which is
either an initial or accepting instantaneous
escription.

The computation relation $\vdash \underline{C}$ ID x ID describes
manner in which the TPDA functions. Given two
tantaneous descriptions id_1 and id_2, $id_1 \vdash id_2$
ad as "id_1 yields id_2") if and only if one of the

following three conditions are met:

(i)

$$id_1 = (\{(s_0, u0, \perp)\} \vee b, t)$$

$$id_2 = (\{(q, u, F)\} \vee b, t)$$

where $b \in 2^{S \times N^* \times T_\Gamma}$, $u \in dom(t)$, $a = t(u)$,

$r(a) = r(F) = 0$, and $(q, F) \in \delta(s_0, a)$.

In other words, this type of move corresp shift-move (or read-move) across the leaf u. operation causes the read-head to be advanced the leaf u, the state is updated to q, and the tree stack is replaced by the one node tree st labeled by F. All other read-heads, and their and tree stacks, referenced by b, are unaltere Graphically, this type of move is depicted in 5.1.9.

(ii)

$$id_1 = (\{(q_i, ui, p_i) \mid 1 \leq i \leq m\} \vee b, t)$$

$$id_2 = (\{(q, u, F(p_1, \ldots, p_m))\} \vee b, t)$$

where $b \in 2^{S \times N^* \times T_\Gamma}$, $u \in dom(t)$, $t(u) = f \in \Sigma$

$r(f) = r(F) = m > 0$, $ui \in dom(t)$ for all i, $1 \leq i \leq m$

$(q, F) \in \delta((q_1, q_2, \ldots, q_m), f)$.

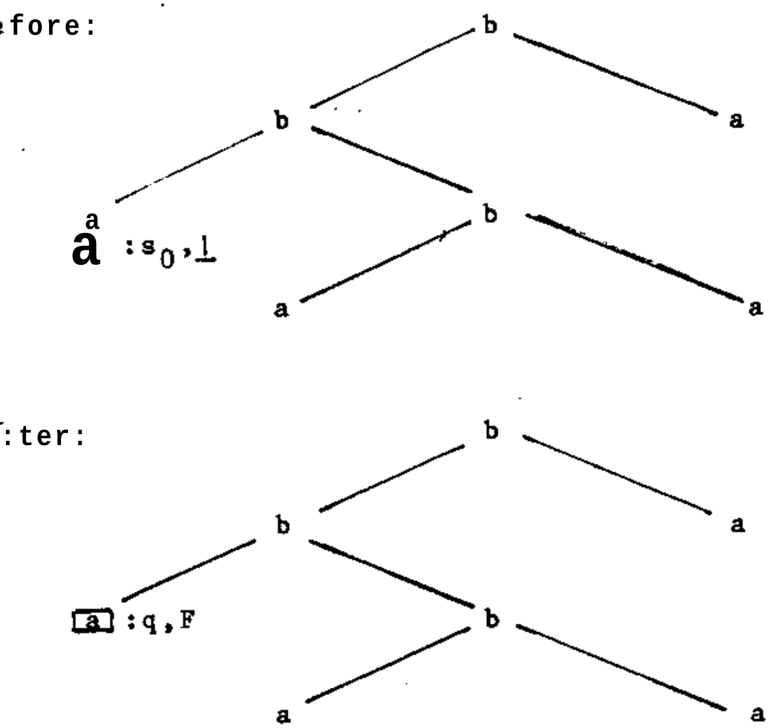


Figure 5.1.9

shift-move over a leaf where (q,F)6δ(SQ,a). Not*
ily the read-head which is being updated has been
town.

This type of move corresponds to a shift-move OA
nternal (non-leaf) node u. Beforehand, there an
-heads located at each of its immediate
endants, and the corresponding states, associatec
i each read head, match those defined by the
tsition map δ (i.e. $q_1, q_2, \cdots, q_m$). After the mo^
•erformed, the read-heads are merged together int<

single read-head covering the node u, the s
updated to q, and the tree stacks are merge
by composing them together with a new root
F. As with the previous form of a shift-mo
other read-heads, and their corresponding s
tree stacks, referenced by b are unaltered.
of move is depicted in figure 5.1.10

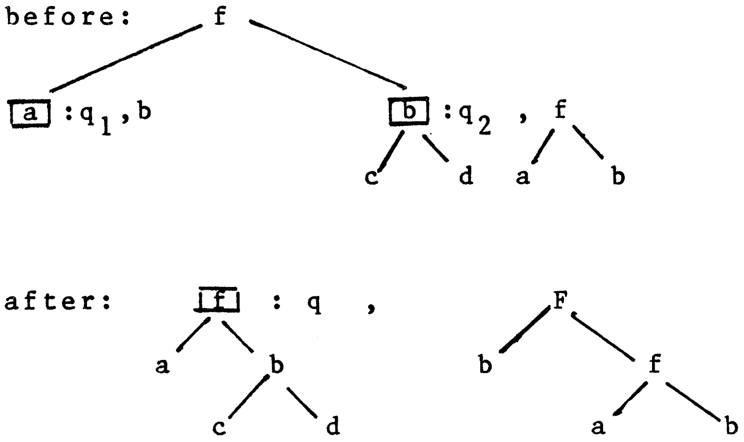


Figure 5.1.10

A shift-move over an internal node where
$(q,F)\in\delta((q_1,q_2),f)$.

ii)

$id_1 = (\{(q_1,u,\beta(t_1,...,t_m))\} \vee b,t)$

$id_2 = (\{(q_2,u,F(t_1,...,t_m))\} \vee b,t)$

where $b \in 2^{S \times N^* \times T_\Gamma}$, $m=r(F)$, any sequence of trees $t_1,...,t_m \in T_\Gamma$ such that for all i, $1 \leq i \leq m$ $t_i(\varepsilon)=F_i$, and $(q_2,F) \in \delta((q_1,\beta[[F_1,...,F_m]]),\varepsilon)$

This type of computation is a reduce-move whe eading takes place. Only the tree stack is modif nd the corresponding state is updated. Note that natching" takes place on the tree stack $\beta(t_1,...,$ verify that it is in the proper form to reduce. her words, for every occurrence of a variable x_i , the corresponding subtree of the tree stack mus atch the tree t_i. Furthermore, this move require nat for each tree t_i, $1 \leq i \leq m$, the root of the tree ust be labeled with the stack symbol F_i. If a le equence of trees $t_1,..,t_m$ is found, which meets t onditions, the tree stack is replaced with the tr $(t_1,...t_m)$ and the state is updated to q_2. raphically, figure 5.1.11 depicts a reduce-move o nis form. Also note that if for some i, $1 \leq i \leq m$, t ariable x_i does not occur in β, then any tree t_i, nose root is labeled with the symbol F_i, can be c i.e. a countable infinity of tree sequences $t_1,..$ ill satisfy the matching condition, thereby causi infinite nondeterminism).

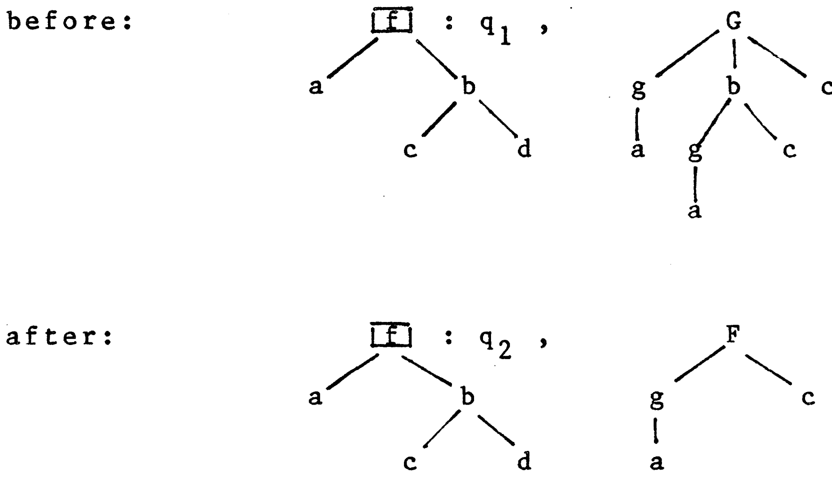
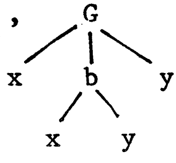
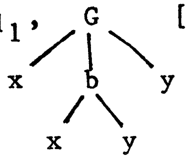


Figure 5.1.11

A reduce-move where $(q_2,F)\in\delta((q_1,$  [[g,

(<u>Note</u>: The above transformation would be ill

the transformation was defined by

$(q_2,F)\in\delta((q_1,$  [[g,g]]),e)

since, for the tree t_2=c, the root is not la

the symbol f)

An input tree t is accepted by a TPDA D if an nly if there exists a computation $id_S \vdash^* id_F$ whe $\vdash^*$ is the transitive reflexive closure of $\vdash$, id_S he initial instantaneous description for the inpu ree t, and id_F is an accepting instantaneous escription of the form $(\{(q,\varepsilon,\perp)\},t)$ where q is a inal state in Q. In other words, the accepting ondition is both final state and empty tree stack et N(D) be the set of all trees accepted by a TPD ence,

$$(D) = \{(t \in T_\Sigma \mid$$
$$id_S = (\{(s_0, u0, \perp) \mid u \in leaf(t)\}, t),$$
$$id_F = (\{(q,\varepsilon,\perp), t),$$
$$q \in Q, \text{ and } id_S \vdash^* id_F\}.$$

xample **5.1.1:** Let $D = (S,\Sigma,\Gamma,0,\perp,Q)$ be a TPDA su hat

$S = \{0,1,2,3,4\}$;

$\Sigma = \{a,f,g\}$ where $r(a)=0$, $r(F)=1$, and $r(G)=2$;

$\Gamma = \{a,f,g,F,\perp\}$ where $r(a)=r(\perp)=0$, $r(f)=r(F)=1$, and $r(g)=2$;

$Q = \{0\}$; and

δ is defined by the following table where for a input pair (a,b), the rows represent possible v of a and the columns represent possible values

Furthermore, empty table entries represent

sets.

	a	f	g	
0	{(1,a)}			
(1)		{(2,f)}		
(1,1)			{(3,g)}	
(3, g [[a]]) / \\ x x				{(
(3, g [[f]]) / \\ x x				{(
(2)		{(2,f)}		
(2,2)			{(3,g)}	
(4, F [[a]]) | f | x				{(
(4, F [[f]]) | f | x				{(
(4, F [[ ]]) | a				{(

The language accepted by D is the set of trees of the form

```
              g
           /     \
          f       f
          |       |
          .       .
   n      .       .      n        where n≥0
          .       .
          |       |
          f       f
          |       |
          a       a
```

For example, the tree

```
    g
  /   \
 f     f
 |     |
 f     f
 |     |
 a     a
```

is accepted as follows:

```
({(0,1110,⊥),(0,2110,⊥)},     g     )  ⊢
                            /   \
                           f     f
                           |     |
                           f     f
                           |     |
                           a     a

({(1,111,a),(0,2110,⊥)},      g     )  ⊢
                            /   \
                           f     f
                           |     |
                           f     f
                           |     |
                           a     a
```

```
({(2,11,f),(0,2110,⊥)},     g     )  ⊢
         |                 /   \
         a                f     f
                          |     |
                          f     f
                          |     |
                          a     a

({(2,1,f),(0,2110,⊥)},     g     )  ⊢
       |                  /   \
       f                 f     f
       |                 |     |
       a                 f     f
                         |     |
                         a     a

({(2,1,f),(1,211,a)},     g     )  ⊢
       |                 /   \
       f                f     f
       |                |     |
       a                f     f
                        |     |
                        a     a

({(2,1,f),(2,21,f)},     g     )  ⊢
       |          |     /
       f          a    f     f
       |               |     |
       a               f     f
                       |     |
                       a     a

({(2,1,f),(2,2,f)},     g     )  ⊢
       |        |     /   \
       f        f    f     f
       |        |    |     |
       a        a    f     f
                     |     |
                     a     a

({(3,ε,    g    )},     g     )  ⊢
          /   \        /   \
         f     f      f     f
         |     |      |     |
         f     f      f     f
```

```
({(4,ε,F)},  g    ) ⊢
       i   /   \
       f   f     f
       i   i     i
       f   f     f
       i   i     i
       a   a     a
({(4,ε,F)},  g    ) ⊢
       I   /   \
       f   f     f
       I   I     I
       a   f     f
           1     I
           a     a
({(4,ε,F)} ,   g   ) ⊢
       I   /   \
       a   f     f
           I     I
           f     f
           I     I
           a     a
({(0,ε,⊣)},  g    )
           /   \
           f     f
           I     I
           f     f
           I     I
           a     a
```

which is the accepting condition.

The meaning of determinism for a TPDA is if is only one possible move (or transition) define each read-head in every legal instantaneous description. In other words, a TPDA D is <u>determ</u> if and only if

i) For all input pairs (a,b), $|\delta(a,b)| \leq$

ii) All states used in reduce-moves are n in shift-moves. That is, if $(q_2,F)\in\delta((q_1,t\,[[F_1,\ldots,F_m]]),\varepsilon)$, then not used in a shift-move.

iii) If $(q_2,F)\in\delta((q_1,\beta\,[[F_1,\ldots,F_m]]),\varepsilon)$ wh $r(F)=m$, then for all i, $1\leq i\leq m$, there $u\in dom(\beta)$ such that $\beta(u)=x_i$.

In other words, condition (i) guarantees that no shift-shift conflicts, condition (ii) guara that there are no shift-reduce conflicts, and conditions (i) and (iii) guarantees that there reduce-reduce conflicts. Also, condition (iii that every reduce-move must be defined on a conservative rule.

5.2 Stateless Tree Pushdown Automata

Quite often, a TPDA will be defined such set of states S will be the same as the set of stack symbols Γ. Furthermore, the current st associated with every read-head will always be symbol labelling the root of the tree stack. there is no need to explicitly include the set ates S in the definition of a TPDA. Whenever thi e case, a TPDA can be simplified to a Stateless t shdown automaton (denoted STPDA) defined by the tuple D = $(\Sigma, \Gamma, \delta, \perp)$ where

Σ is a finite ranked alphabet of input symbols;

Γ is a finite ranked alphabet of stack symbols;

δ : $(\{\perp\} \times \Sigma)$

$\quad \vee$ (tuples(Γ) x Σ)

$\quad \vee$ ($T_{\Gamma}(X_m)$ x tuples(Γ)) x $\{\varepsilon\}$) -> 2^{Γ}

is a function called the transition map where $m=\max\{r(\beta) \mid \beta\in\Gamma\}$, and δ has finite domain;

$\perp\in\Gamma$ is a reserved constant denoting the empty tree stack.

Furthermore, the transition map δ is simplifie om the TPDA such that all definitions are one of llowing two forms:

ift-moves:

i) $F\in\delta(\perp,a)$ where $a\in\Sigma$, $F\in\Gamma$, and $r(a)=r(F)=0$

ii) $F\in\delta((q_1,\ldots,a_m),f)$ where $f\in\Sigma$, $F\in\Gamma$, $r(f)=r(F)=m>0$, and $q_1,\ldots,q_m\in\Gamma$

Reduce-move:

$F \in \delta((t,(F_1,\ldots,F_m)),\varepsilon)$ where $F \in \Gamma$, $m=r(F)$

$t \in T_\Gamma(X_m)$, and $F_1,\ldots,F_m \in \Gamma$.

As before, to emphasize the look-back nature o

$(F_1,\ldots,F_m)$, $\delta((t,(F_1,\ldots,F_m)),\varepsilon)$ will be deno

$\delta(t\ [[F_1,\ldots,F_m\ ]],\varepsilon)$.

The <u>instantaneous description of a STPDA</u>

SID) is a pair $(a,t) \in 2^{N^* \times T_\Gamma} \times T_\Sigma$ where t is

tree, and a is a set of pairs (u,p) where u is

address of a node covered by a read-head, and

tree stack associated with that read-head. Al

initial configuration becomes the SID

$(\{(u0,\perp) \mid u \in leaf(t)\},t)$.

The computation relation $\vdash \underline{C}$ SID x SID

such that $id_1 \vdash id_2$ if and only if one of th

following three conditions hold:

i) $id_1 = (\{(u0,\perp)\} \vee b,t)$

$id_2 = (\{(u,F)\} \vee b,t)$

where $b \in 2^{N^* \times T_\Gamma}$, $u \in dom(t)$, $t(u)=a \in \Sigma$

$r(a)=r(F)=0$, and $F \in \delta(\perp,a)$

$id_1 = (\{(ui,p_i) \mid 1 \leq i \leq m\} \vee b, t)$

$id_2 = (\{(u,F(p_1,...,p_m))\} \vee b, t)$

where $b \in 2^{N^*} \times T_\Gamma$, $u \in dom(t)$, $t(u)=f \in \Sigma$, $F \in \Gamma$, $r(f)=r(F)=m>0$, $ui \in dom(t)$ for all i, $1 \leq i \leq m$, and $F \in \delta((q_1,...,q_m),f)$ such that for all i, $1 \leq i \leq m$, $p_i(\varepsilon)=q_i$.

$id_1 = (\{(u,\beta(t_1,...,t_m))\} \vee b, t)$

$id_2 = (\{(u,F(t_1,...,t_m)\} \vee b, t)$

where $b \in 2^{N^*} \times T_\Gamma$, $F \in \Gamma$, $r(F)=m$, any sequence of trees $t_1,...,t_m \in T_\Gamma$ such that for all i, $1 \leq i \leq m$, $t_i(\varepsilon)=F_i$, and $F \in \delta(\beta\ [[F_1,...,F_m]],\varepsilon)$.

the computations for a STPDA are identical to a TPDA, with the exception that the explicit s to states have been removed. Graphical of each of these three types of moves are in figures 5.2.1 - 5.2.3

lly, an input tree t is accepted by a STPDA if if there is a computation $id_S \vdash^* id_F$ where e initial SID for the input tree t, and id_F is rm $(\{(\varepsilon,\perp)\},t)$. More formally, the <u>tree</u> <u>accepted</u> <u>by</u> <u>a</u> <u>STPDA</u> <u>D</u> (denoted N(D)) is the

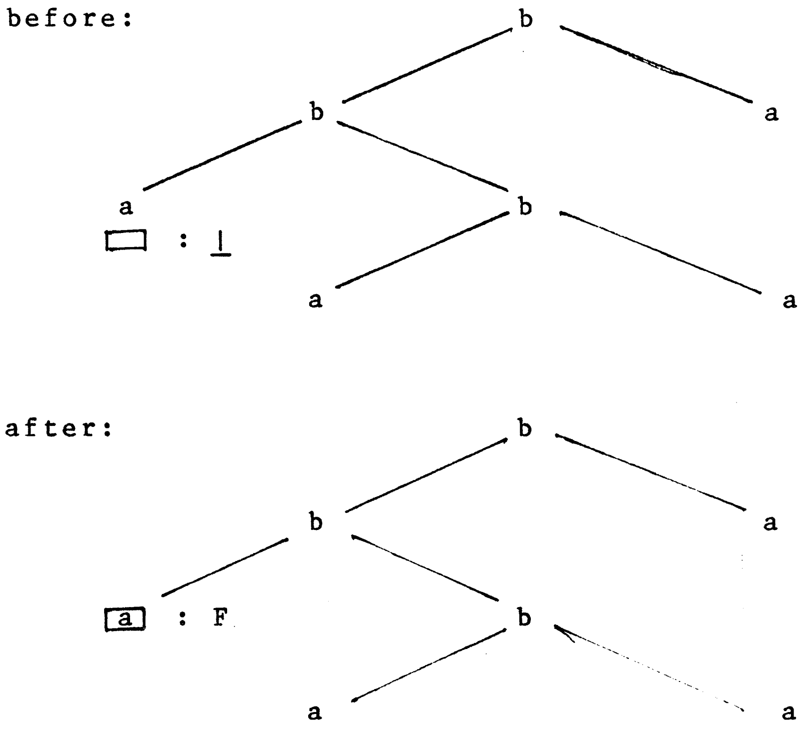


Figure 5.2.1

A shift-move over a leaf where $F \in \delta(\underline{\perp}, a)$. <u>Note</u>: only the read-head which is being updated has b
shown.

:

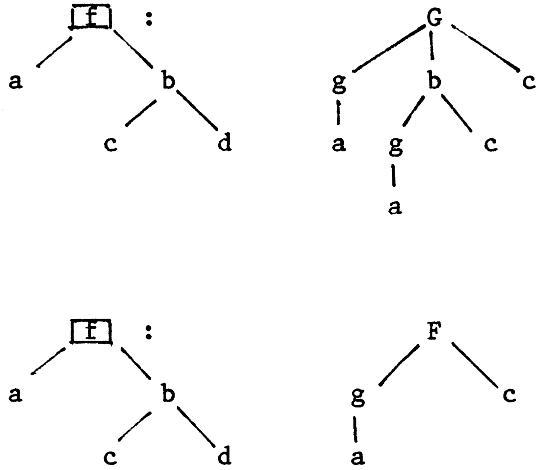
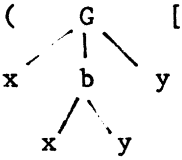
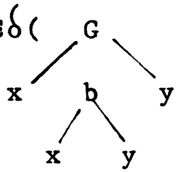


Figure 5.2.3

:e-move where $F \in \delta($  $[[g,c]]),e)$.

The above transformation would be illegal if
ansformation was defined by

$F \in \delta($  $[[g,g]]),e)$

for the tree $t_2=c$, the root is not labeled by
abol f)

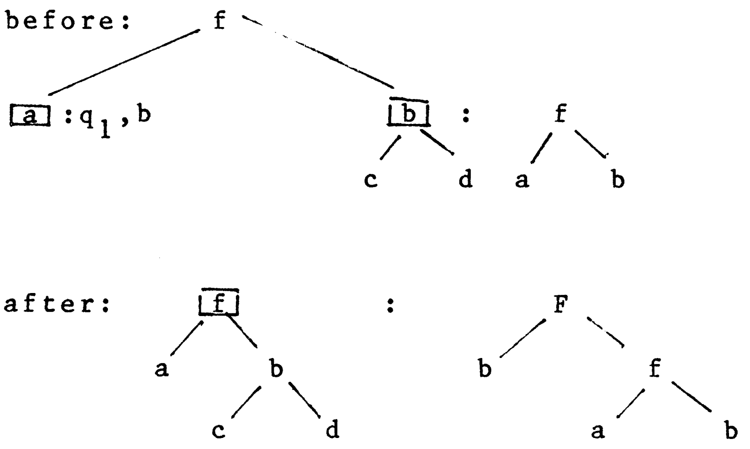


Figure 5.2.2

A shift-move over an internal node where $F \in \delta((q_1,q_2),f)$.

$$(D) = \{(t \in T_{\overline{\Sigma}} \mid$$
$$id_S = (\{(u0,\perp) \mid u \in leaf(t)\},t),$$
$$id_F = (\{(\varepsilon,\perp),t),$$
$$\text{and } id_S \vdash^{*} id_F\}.$$

xample 5.2.1: Let D be the TPDA defined in exampl .1.1. By converting the set of states S such tha orresponds to $\perp$, 1 corresponds to a, 2 correspond , 3 corresponds to g, and 4 corresponds to F, the f states S can be mapped bijectively to $\overline{\Gamma}$. Henc an be simplified to a STPDA D′ such that

′ = $(\overline{\Sigma},\overline{\Gamma},\delta,\perp)$ where $\overline{\Sigma}$ and $\overline{\Gamma}$ are defined as befo is defined as follows:

	a	f	g	6
1	{a}			
(a)		{f}		
>,a)			{g}	
[[a]] c				{F}
Kf 11 c				{F}
(f)		{f>		
:f,f)			{g}	
[[a]]				{F}
[[f]]				{F}
[[ ]]				{1}

ample, the tree g is

```
     g
   /   \
  f     f
  |     I
  f     f
  I     I
  a     a
```

ad as follows:

```
({(1110,⊥),(2110,⊥)},      g     )  ⊢
                          /   \
                         f     f
                         |     |
                         f     f
                         |     |
                         a     a

({(111,a),(2110,⊥)},      g     )  ⊢
                         /   \
                        f     f
                        |     |
                        f     f
                        |     |
                        a     a

({(11,f),(2110,⊥)},      g     )  ⊢
      |                 /   \
      a                f     f
                       |     |
                       f     f
                       |     |
                       a     a

({(1,f),(2110,⊥)},      g     )  ⊢
     |                 /   \
     f                f     f
     |                |     |
     a                f     f
                      |     |
                      a     a

({(1,f),(211,a)},      g     )  ⊢
     |                /   \
     f               f     f
     |               |     |
     a               f     f
                     |     |
                     a     a

({(1,f),(21,f)},      g     )  ⊢
     |       |       /   \
     f       a      f     f
     |              |     |
     a              f     f
```

```
f),(2,f)},   g   )  ⊢
|     |    /   \
f     f   f     f
|     |   |     |
a     a   f     f
          |     |
          a     a

   g   )},   g   )  ⊢
 /   \     /   \
f     f   f     f
|     |   |     |
f     f   f     f
|     |   |     |
a     a   a     a

F)},   g   )  ⊢
|    /   \
f   f     f
|   |     |
f   f     f
|   |     |
a   a     a

F)},   g   )  ⊢
|    /   \
f   f     f
|   |     |
a   f     f
    |     |
    a     a

F)},   g   )  ⊢
|    /   \
a   f     f
    |     |
    f     f
    |     |
    a     a
```

```
({(ε,⊥)},    g    )
           /   \
          f     f
          |     |
          f     f
          |     |
          a     a
```

which is an accepting condition.

Like a TPDA, a STPDA is <u>deterministic</u> if

if

i) for all input pairs (a,b), $|\delta(a,b)|$

ii) If $F \in \delta(t\ [[F_1,\ldots,F_m]],\varepsilon)$ where $r(F)=$

$G=t(t_1,\ldots,t_m)(\varepsilon)$ for any sequence o

$t_1,\ldots,t_m \in T_\Gamma$ where $t_i(\varepsilon)=F_i$ for all

then G is not used in a shift move.

iii) if $F \in \delta(\beta\ [[F_1,\ldots,F_m]],\varepsilon)$ where $r(F)=$

for all i, $1 \leq i \leq m$, there exists a $u \in$ d

that $\beta(u)=x_i$.

In other words, there are no shift-shift, shi

or reduce-reduce conflicts.

Equivalence To Tree Grammars

The purpose of this section is to provide proofs
how that the class of tree languages generated by
grammars (under an OI derivation), the class of
languages accepted by TPDAs, and the class of tr
uages accepted by STPDAs are identical. These
lts are shown in four steps. Section 5.3.1 shows
every tree grammar (in weak Chomsky normal form)
be converted to some STPDA. Section 5.3.2 shows
every STPDA can be converted to some TPDA. To
lete the circle, section 5.3.3 shows that every
can be converted to some tree grammar. Finally,
ion 5.3.4 uses these results to show that the thr
ses of tree languages are identical.

However, before showing these results, this
ions starts by introducing an ordering on the
utation relation (in the same manner as done with
erivations in chapter 3). The notions of a
utation with a postfix lower bound u and a
utation under a postfix ordering, are introduced.
hermore, it will be shown that any computation ca
onverted to a computation under a postfix orderi

To simplify the notation, let an *updated*

of a computation $id_1 \vdash id_2$ (denoted URH(id_1

be a triple of the form $(q,u,p) \in S \times \mathbf{N}^* \times T_\Gamma$ w

$id_1, id_2 \in ID$; $id_1 \vdash id_2$; and id_1, id_2 are in

following three forms:

i) $id_1 = (\{(s_0, u0, \perp)\} \vee b, t)$ and

$id_2 = (\{(q,u,F)\} \vee b, t)$

where $r(t(u))=0$ and $p=F$

ii) $id_1 = (\{(q_i, ui, p_i) \mid 1 \leq i \leq m\} \vee b, t)$ and

$id_2 = (\{(q,u,F(p_1,\ldots,p_m)\} \vee b, t)$

where $r(t(u))=m>0$ and $p=F(p_1,\ldots,p_m)$

iii) $id_1 = (\{(q', u, \beta(t_1,\ldots,t_m))\} \vee b, t)$ an

$id_2 = (\{(q,u,F(t_1,\ldots,t_m))\} \vee b, t)$

where $m=r(F)$ and $p=F(t_1,\ldots,t_m)$

A *computation with postfix lower bound u*

TPDA) is the relation $\vdash^u$: ID x ID, in which

computation can be performed in either the su

or in some subtree to the "right" of t/u. Le

id_1, id_2 be two instantaneous descriptions, id

if and only if $id_1 \vdash id_2$, URH($id_1 \vdash id_2$)=(q

some $(q,v,p) \in S \times \mathbf{N}^* \times T_\Gamma$, and $u \leq v$ where the r

is the postfix ordering.

Similarly, a <u>computation under a postfix orderi</u>
any computation such that the updated read-heads
n computation step are sorted by a postfix orderi
other words, given the computation

$id_1 \vdash id_2 \vdash \ldots \vdash id_n$ for any $n \geq 1$,

i) for each i, $1 \leq i \leq m$,
$URH(id_i \vdash id_{i+1}) = (q_i, u_i, p_i)$ for some $(q_i, u_i, p_i) \in S \times \mathbf{N}^* \times T_{\Gamma}$

ii) for all i, $1 \leq i \leq m$, for all j, $i \leq j \leq m$, $u_i \leq u_j$
where $\leq$ is the postfix ordering,

n $id_1 \vdash id_2 \vdash \ldots \vdash id_n$ is a computation unde
tfix ordering. Whenever a computation $id_1 \vdash id_2$
$\vdash id_n$ is a computation under a postfix orderin
will be denoted as $id_1 \vdash^1 id_2 \vdash^1 \ldots \vdash^1 id_n$.

Using these definitions, it is possible to show
t computations can be commuted (to some extent)
never they are applied to independent subtrees.
s is shown by the following lemma.

<u>ma 5.3.1:</u> Given a TPDA $D=(S,\Sigma,\Gamma,\delta,s_0,\perp,Q)$, any t
tantaneous descriptions $id_1, id_2, id_3 \in ID$, for any
*, any $n \geq 0$, if $id_1 \vdash^{u\ n} id_2 \vdash id_3$ where
$(id_2 \vdash id_3) = (q,v,p)$ for some $(q,v,p) \in S \times \mathbf{N}^* \times$

and $v<u$ (under a postfix ordering), then there

an instantaneous description id_2' such that

$id_1 \vdash id_2' \vdash^{u\ n} id_3$ where $URH(id_1 \vdash id_2')=(q,$

Proof: By induction.

<u>base case:</u> $id_1 \vdash id_2$. Trivial.

<u>inductive step:</u> $id_1 \vdash^u id_2 \vdash^{u\ n} id_3 \vdash id_4$ su

$URH(id_1 \vdash id_2)=(q_2,u_2,p_2)$, $URH(id_3 \vdash id_4)=($

and $u_4<u\leq u_2$. By induction, $id_1 \vdash^u id_2 \vdash id_3'$

such that $URH(id_2 \vdash id_3')=(q_4,u_4,p_4)$. By the

definition of $\vdash$,

$$id_2 = (b_1 \vee b_2 \ , \ t)$$

$$id_3' = (\{(q_4,u_4,p_4)\} \vee b_2 \ , \ t)$$

where b_1 is one of the following forms:

i) $\{(s_0,u_4 0,\bot)\}$ where $r(t(u_4))=0$

ii) $\{(q_4^i,u_4 i,p_4^i) \mid 1\leq i\leq m\}$ where $r(t(u_4))=$

iii) $\{(q_4',u_4,p_4')\}$.

By the definition of $\vdash^u$,

$$id_1 = (b_1 \vee b_3 \vee b_4 \ , \ t)$$

$$id_2 = (b_1 \vee \{(q_2,u_2,p_2)\} \vee b_4 \ , \ t)$$

where $\{(q_2,u_2,p_2)\} \vee b_4 = b_2$, and

b_3 is one of the following three forms:

i) $\{(s_Q,u_2 0,\_L)\}$ where $r(t(u_2>)=0$

ii) $\{(q_2^i,u_2 i,p_2^i) \mid 1 \leq i \leq m\}$ where $r(t(u_2))=m>0$

iii) $\{(q'_2,u_2,P'_2)\}$.

nee $u_1 < u_0$, and u_1 is not a prefix of u_0, clearly

$_1 \vdash id'_2 \vdash^U id$^ where $id'_{22} = (\{(q_4,u_4,p_4)\} \vee b_3 \vee b_4$

nee, $id_1 \vdash id'_2 \vdash^u {}^n id_4$.

mma 5.3.2; Given a TPDA D=(S,2,P,6,s ,| ,Q), any

stantaneous descriptions $id_{1f} id$^€ID, $id_1 \vdash^n id_2$

d only if $id_1 \vdash^{1\ n} id_2$.

oof: By induction.

se cases: n^O and n=l. Both are trivial.

ductive step: assume $id_1 \vdash id_2 \vdash^n id_3$ where n;

d URH($id_1 \vdash id_2$)=(q_2,u_2,p_2). By induction,

$_1 \vdash id_2 \vdash^1 id_4 \vdash^{1\ n-1} id_3$ where

H($id_2 \vdash id_4$)=(q_4,u_4,p_4)• Depending on whether c

t u_4^u_2, there are two cases:

se 1 i \i~Ku, . By the definition of $\vdash^1$, clearly

$1 \wedge id_2 \wedge^n id_3$'

i± 1± $^U 4<^U 2$* By lemma 5.3.1, $id_1 \vdash id'_2 K^U id_4$

at URH($id_1 \vdash id_p$)=(q_4,u_4,p_4) and

H($id'_2 \wedge id_4$)=(q_2,u_2,p_2). By the definition of

early $id_1 \vdash^1 id'_2 \vdash^{1\ n} id_3$.

erefore $id_1 \vdash^{1\ n+1} id_3$.

To show that the other direction, assume $id_1 \vdash^{1\ n+1} id_2$. Clearly, by the definition o must be the case that $id_1 \vdash^{n+1} id_2$.

The same reasoning, as above, can be use STPDAs. The notions of a computation with pos lower bound u, as well as a computation under ordering are introduced.

Let an <u>updated read-head of a computatio</u> $id_1 \vdash id_2$ for a STPDA (denoted URH($id_1 \vdash id$ pair of the form $(u,p) \in \mathbf{N}^* \times T_{\Gamma}$ where $id_1, id_2 \in$ $id_1 \vdash id_2$; and id_1, id_2 are in one of the fo three forms:

i) $id_1 = (\{(u0,\underline{|})\} \vee b,\ t)$ and

$id_2 = (\{(u,F)\} \vee b,\ t)$ where $p=F$

ii) $id_1 = (\{(ui,p_i) \mid 1 \leq i \leq m\} \vee b,\ t)$ and

$id_2 = (\{(u,F(p_1,\ldots,p_m))\} \vee b,\ t)$ wh

$r(t(u))=m>0$ and $p=F(p_1,\ldots,p_m)$

iii) $id_1 = (\{(u,\beta(t_1,\ldots,t_m))\} \vee b,\ t)$ an

$id_2 = (\{(u,F(t_1,\ldots,t_m))\} \vee b,\ t)$ wh

$r(F)=m$ and $p=F(t_1,\ldots,t_m)$.

computation with postfix lower bound u for a
s the relation $\vdash^u \subseteq$ SID x SID defined such that
two stateless instantaneous descriptions
$\in$SID, $id_1 \vdash^u id_2$ if and only if $id_1 \vdash id_2$ and
$\vdash id_2)=(v,p)$ for some $(v,p)\in\mathbf{N}^* \times T_{\Gamma}$ where $u \leq v$
postfix ordering relation $\leq$.

milarly, a computation under a postfix ordering
TPDA is defined such that for any computation
$id_2 \vdash \ldots \vdash id_n$, if

for all i, $1 \leq i \leq n$, $URH(id_i \vdash id_{i+1})=(u_i,p_i)$
for some $u_i \in \mathbf{N}^*$ and some $p_i \in T_{\Gamma}$

for all i, $1 \leq i \leq n$, for all j, $i \leq j \leq n$, $u_i \leq u_j$
where $\leq$ is the postfix ordering relation

$d_1 \vdash id_2 \vdash \ldots \vdash id_n$ is a computation under
ix ordering. Also, whenever a computation id_1
$\vdash \ldots \vdash id_n$ is a computation under a postfix
g, it will be denoted as $id_1 \vdash^1 id_2 \vdash^1 \ldots \vdash^1$

ing these definitions, the following lemmas are
ed.

Lemma **5.3.3:** Given a STPDA $D=(\Sigma,\Gamma,\delta,\perp)$, any three stateless instantaneous descriptions $id_1,id_2,id_3 \in$ for any $u \in \mathbf{N}^*$, any $n \geq 0$, if $id_1 \vdash^{u\ n} id_2 \vdash id_3$ whe JRH$(id_2 \vdash id_3)=(v,p)$ for some $(v,p) \in \mathbf{N}^* \times T_\Gamma$ and (under a postfix ordering), then there exists a stateless instantaneous description id_2' such that JRH$(id_1 \vdash id_2')=(v,p)$.

Proof: Analogous to lemma 5.1.1.

Lemma **5.3.4:** Given a STPDA $D=(\Sigma,\Gamma,\delta,\perp)$, any two stateless instantaneous descriptions $id_1,id_2 \in SID$, $id_1 \vdash^{n} id_2$ if and only if $id_1 \vdash^{1\ n} id_2$.

Proof: Analogous to lemma 5.3.2.

Since every computation can be converted to computation under a postfix ordering, the remaind this thesis will assume that all computations wil under a postfix ordering.

.1 Converting Tree Grammars Into STPDAs -

This section shows that every tree grammar G c converted to a STPDA D such that $L_{OI}(G) = N(D)$. a used in this conversion resembles the conversi d to convert a string grammar in Chomsky Normal a PDA where the moves of the PDA simulate deriva ps and hence, the PDA accepts a string α if and it can simulate the derivation that produced the ing α (see Harrison[78], Lewis & Papadimitriou[utzenberger[63], Chomsky[62], and Evey[63]). ever, due to the nature of the definition of a T e. a bottom-up parser instead of a top-down par method presented in this section simulates the erse of derivation steps and hence, simulates ivations in reverse.

As mentioned above, the idea used in the version of a tree grammar to a TPDA is to have e uce-move act as the inverse operation of an OI ivation-step. Hence, the set of stack symbols i set of nonterminals in the tree grammar. thermore, the conversion maintains the property every read-head, if the subtree the read-head nned is replaced by the tree stack associated wi t read-head, and the input tree is in the tree

language generated by the tree grammar, the res
tree is a sentential form. However, in order t
nonconservative productions, the set of product
also added to the set of stack symbols where a
production is an intermediate stack symbol used
simulation of derivation steps.

Definition 5.3.1: Given a tree grammar $G=(\Phi,\Sigma,P$
weak Chomsky normal form, let the corresponding
$D=(\Sigma,\Gamma,\delta,\perp)$ where

$\Gamma = \Phi \cup P \cup \{\perp\}$ where $\perp \notin \Phi$, $r(\perp)=0$, and for each production $F(\vec{x})\text{->}t \in P$, $r(F(\vec{x})\text{->}t)=r(t(\varepsilon))$; a

δ is defined by the following four conditions:

(1) $\perp \in \delta(S\ [[\ ]],\varepsilon)$

This condition states that if the input tree is
derivable from the start production, then the i
tree should be accepted.

(2) if $p=F(\vec{x})\text{->}a \in P$ where $a \in \Sigma$ and $r(a)=0$, then

a) $p \in \delta(\perp, a)$

b) for every tuple $(G_1,\ldots,G_m) \in tuple_m(\Phi)$
$r(F)=m$, $F \in \delta(p\ [[G_1,\ldots,G_m\ ]],\varepsilon)$

This condition simulates a derivation step of t
$F(t_1,\ldots,t_m) \underset{OI}{\Longrightarrow} a$ using the production $F(\vec{x})\text{->}a$

erse of the derivation is simulated using two
putation moves as follows: First, a leaf of the
ut tree labeled by the terminal symbol "a" is rea
ng a read-move where the tree stack gets updated
one node tree labeled with the production $F(\vec{x})\text{->}$
ng the transition $(F(\vec{x})\text{->}a)\in\delta(\underline{\perp},a)$. Then, the on
e tree stack is updated to the tree stack
$_1,...,t_m)$ by performing a reduction using the
nsition $F\in\delta((F(\vec{x})\text{->}a)\ [[G_1,...,G_m]],\varepsilon)$ where for a
$1\leq i\leq m$, $t_i(\varepsilon)=G_i$ (see figure 5.3.1). One should n
t if $F(\vec{x})\text{->}a$ is a conservative production, the ab
cess could have been done in a single transition.
ever, instead of making separate conditions for
servative and nonconservative productions, both w
dled using two computation moves.

$$\begin{array}{lllll} a & \vdash & \boxed{a} : F\text{->}a & \vdash & \boxed{a} : F \\ \square : \bot & & & & \end{array}$$

(i) The corresponding computation for the

derivation step $F \Longrightarrow a$ where $F\text{->}a \in P$ is a

conservative production.

$$\begin{array}{lllll} a & \vdash & \boxed{a} : F(\vec{x})\text{->}a & \vdash & \boxed{a} : \\ \square : g & & & & \\ & & & & t_1 \end{array}$$

(ii) The corresponding computation for th

derivation step $F(t_1,\ldots,t_m) \Longrightarrow a$ where F

is a nonconservative production.

Figure 5.3.1: simulation of a

derivation step using the production $F(\vec{x}$

(3) if $p=F(\vec{x})\text{->}f(x'_1,\ldots,x'_q) \in P$ where $f \in \overline{\Sigma}$, $r(f)=$

$r(F)=m$, and for all i, $1 \le i \le q$, $x'_i \in \mathbf{X}_m$, then

a) for each $(G_1,\ldots,G_q) \in tuple_q(\overline{\Phi})$,

$p \in \delta((G_1,\ldots,G_q),f)$

b) for each $(G_1,\ldots,G_m) \in tuple_m(\overline{\Phi})$,

$F \in \delta(p(x'_1,\ldots,x'_q)\ [[G_1,\ldots,G_m]],\varepsilon)$

This condition simulates a derivation step of

$F(t_1,\ldots,t_m) \Longrightarrow f(t'_1,\ldots,t'_q)$ using the produ

$>f(x_1', \ldots, x_q')$ where $r(f)=q$ and for all i, $1 \leq i \leq q$,
$=x_j$, then $t_i'=t_j$. Like the previous condition, th
ation step is simulated by performing two
tation moves. First, the node labeled by f is
using the read-move defined by the transition
$\to f(x_1', \ldots, x_q')) \in \delta((G_1, \ldots, G_q), f)$ where for all i,
, $t_i'(\epsilon)=G_i$. Then, the second step uses a
e-move to introduce each tree
$\{t_1, \ldots, t_m\}-\{t_1', \ldots, t_q'\}$ due to the fact that the
ction $F(Tf)\to f(x_1', \ldots, x_q')$ may not be conservative
ermutes these trees to the tuple $(t_1, \ldots, t_m)$ (see
e 5*3.2 below).

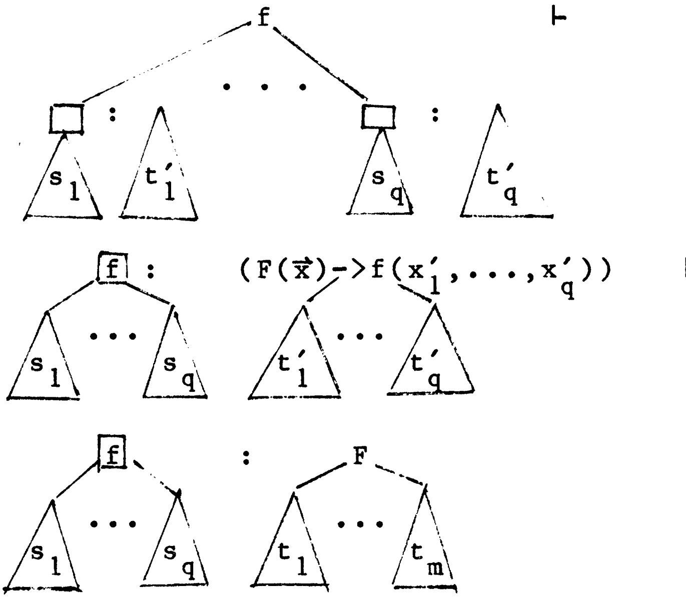


Figure 5.3.2: Simulation of the derivati

$F(t_1,\ldots,t_m) \Longrightarrow f(t'_1,\ldots,t'_q)$ where $f(t'_1$

$\Longrightarrow^* f(s_1,\ldots,s_q)$.

(4) if $F(\vec{x}) \to t \in P$ where $r(F)=m$ and $t \in T_{\bar{\Phi}}(X_m)$, t

all tuples $(G_1,\ldots,G_m) \in tuple_m(\bar{\Phi})$,

$F \in \delta(t\ [[G_1,\ldots,G_m\ ]],\varepsilon)$.

This condition simulates a derivation step of

$F(t_1,\ldots,t_m) \overset{}{\underset{OI}{\Longrightarrow}} t(t_1,\ldots,t_m)$ where t is a tr

by nonterminals and variables. The derivatio

simulated with a single reduce-move and is de

figure 5.3.3.

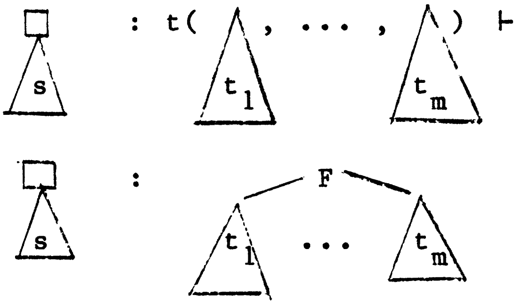


Figure 5.3.3: Simulation of the derivation step $F(t_1,\ldots,t_m) \Longrightarrow t(t_1,\ldots,t_m)$ where $t \in T_{\bar{\Phi}}(X_m)$ an $t(t_1,\ldots,t_m) \Longrightarrow^{*} s$.

nple 5.3.1: Let $G=(\bar{\Phi},\bar{\Sigma},P,S)$ where

$\bar{\Phi} = \{S,F,\hat{a},\hat{f},\hat{g}\}$ where $r(S)=r(\hat{a})=0$,

$r(F)=r(\hat{f})=1$, and $r(\hat{g})=2$;

$\bar{\Sigma} = \{a,f,g\}$ where $r(a)=0$, $r(f)=1$,

and $r(g)=2$; and

P is the set of productions

```
p1: S -> F     p2: F -> F     p3: F -> ĝ
         |           |    |           |    / \
         â           x    f̂           x   x   x
                          |
                          x

p4: â -> a     p5: f̂ -> f     p6: ĝ   ->   g
                    |    |         / \     / \
                    x    x        x   y   x   y
```

te: the tree grammar G generates the same tree guage as in example 5.1.1). The corresponding DA, as defined in definition 5.3.1, is $D=(\bar{\Sigma},\Gamma,\delta,$

where δ is defined as follows:

$$\delta(S\ [[\ ]],\varepsilon) = \{\perp\}$$

$$\delta(\perp,a) = \{p_4\}$$

$$\delta(p_4\ [[\ ]],e) = \{\hat{a}\}$$

$$\delta((S),f) = \delta((F),f) = \delta((\hat{a}),f)$$

$$= \delta((\hat{f}),f) = \delta((\hat{g}),f) = \{p_5\}$$

$$\delta(\underset{\substack{|\\ x}}{p_5}\ [[S\ ]],\varepsilon) = \delta(\underset{\substack{|\\ x}}{p_5}\ [[F\ ]],\varepsilon) = \delta(\underset{\substack{|\\ x}}{p_5}\ [[a\ ]],\varepsilon)$$

$$= \delta(\underset{\substack{|\\ x}}{p_5}\ [[\hat{f}\ ]],\varepsilon) = \delta(\underset{\substack{|\\ x}}{p_5}\ [[\hat{g}\ ]],\varepsilon) = \{\ \underset{\substack{|\\ x}}{\hat{f}}\ \}$$

$$\delta((S,S),g) = \delta((S,F),g) = \delta((S,\hat{a}),g)$$

$$= \delta((S,\hat{f}),g) = \delta((S,\hat{g}),g) = \delta((F,S),g)$$

$$= \delta((F,F),g) = \ldots = \delta((\hat{g},\hat{g}),g) = \{p_6\}$$

$$\delta(\underset{\substack{/\ \ \backslash\\ x\quad y}}{p_6}\ [[S,S\ ]],\varepsilon) = \delta(\underset{\substack{/\ \ \backslash\\ x\quad y}}{p_6}\ [[S,F\ ]],\varepsilon) = \delta(\underset{\substack{/\ \ \backslash\\ x\quad y}}{p_6}\ [[S,\hat{a}\ ]],$$

$$= \delta(\underset{\substack{/\ \ \backslash\\ x\quad y}}{p_6}\ [[S,\hat{f}\ ]],\varepsilon) = \ldots = \delta(\underset{\substack{/\ \ \backslash\\ x\quad y}}{p_6}\ [[\hat{g},\hat{g}\ ]],\varepsilon) = \{\ \ x$$

$$\delta(\underset{\substack{|\\ \hat{a}}}{F}\ [[\ ]],\varepsilon) = \{S\}$$

$$\delta(\underset{\substack{|\\ \hat{f}\\ |\\ x}}{F}\ [[S\ ]],\varepsilon) = \delta(\underset{\substack{|\\ \hat{f}\\ |\\ x}}{F}\ [[F\ ]],\varepsilon) = \ldots = \delta(\underset{\substack{|\\ \hat{f}\\ |\\ x}}{F}\ [[\hat{g}\ ]],\varepsilon)$$

```
δ(ĝ [[S ]],ε)  =  δ(ĝ [[F ]],ε)  =  ...  =  δ(ĝ [[ĝ ]],ε)  =  {
 / \               / \                        / \
x   x             x   x                      x   x
```

example, the derivation

```
S ==OI==> F ==OI==>  F  ==OI==>   ĝ   ==OI==>   g   ==OI==>
          |          |           / \           / \
          â          f̂          f̂   f̂         f̂   f̂
                     |          |   |         |   |
                     â          â   â         â   â

   g    ==OI==>   g   ==OI==>   g   ==OI==>   g
  / \            / \           / \           / \
 f   f̂          f   f̂         f   f         f   f
 |   |          |   |         |   |         |   |
 â   â          a   â         a   â         a   a
```

simulated by the STPDA D as follows:

```
({(110,⊥) , (210,⊥)},    g   )  ⊢ 2
                        / \
                       f   f
                       |   |
                       a   a

({(110,⊥) , (21,â)},    g   )  ⊢ 2
                       / \
                      f   f
                      |   |
                      a   a

({(110,⊥) , (2,f̂)},    g   )  ⊢ 2
                |      / \
                â     f   f
                      |   |
                      a   a

({(11,â) , (2,f̂)},    g   )  ⊢ 2
               |      / \
               â     f   f
                     |   |
                     a   a

({(1,f̂) , (2,f̂)},    g   )   ⊢ 2
     |        |      / \
     â        â     f   f
                    |   |
```

```
({(e,  ĝ  )},  g  ) ⊢
      / \     / \
     f̂   f̂   f   f
     |   |   |   |
     â   â   a   a

({(e, F )},  g  ) ⊢
      |     / \
      f̂    f   f
      |    |   |
      â    a   a

({(e, F )},  g  ) ⊢
      |     / \
      â    f   f
           |   |
           a   a

({(e, S )},  g  ) ⊢
            / \
           f   f
           |   |
           a   a

({(e,⊥)},  g  )
          / \
         f   f
         |   |
         a   a
```

Lemma 5.3.5, lemma 5.3.6, and theorem 5.3. (below) show that for any tree grammar G in WCN the corresponding STPDA D defined by definition $N(D) = L_{OI}(G)$. Lemma 5.3.5 shows that for any $t_1 \in T_{\Phi}$ where $t_1 \overset{*}{\underset{OI}{\Longrightarrow}} t_2 \in T_{\Sigma}$, then for any tree such that $s=s[u \leftarrow t_2]$, $id_1 \vdash^{*} id_2$ where id_1 is initial instantaneous description for the tree the updated read-head of the last computation i

$u,t_1)$. Lemma 5.3.6 shows the converse of lemma 5
y showing that for any computation $id_1 \vdash^n id_2$ wh
d_1 is the initial instantaneous description and t
ast computation produces the updated read-head (u
$\overset{*}{\underset{OI}{\Longrightarrow}}$ s/u where s is the input tree. Finally, th
.3.1 uses the results of these two lemmas to show
esired result that $N(D) = L_{OI}(G)$.

<u>emma</u> <u>5.3.5:</u> Given a tree grammar $G=(\Phi,\Sigma,P,S)$ in w
homsky normal form and the corresponding STPDA
$=(\Sigma,\Gamma,\delta,\perp)$ as defined in definition 5.3.1; any
$\in 2^{N^*} \times T_\Gamma$; any three trees $t_1,t_2,t_3 \in T_\Sigma$ such tha
$_1=t_3[u\text{<-}t_2]$ for some $u \in dom(t_3)$; any $F \in \Phi$ where r(
nd $F(\vec{x})\text{->}t \in P$; any sequence of trees $s_1,\ldots,s_m \in T_\Phi$
nd any $n \geq 0$; if $F(s_1,\ldots,s_m) \underset{OI}{\Longrightarrow} t(s_1,\ldots,s_m) \overset{n}{\underset{OI}{\Longrightarrow}}$
hen $(\{(uv0,\perp) \mid v \in leaf(t_2)\} \vee b, t_1) \vdash^*$
$\{(u,F(s_1,\ldots,s_m))\} \vee b, t_1)$.

roof: By induction on n.

<u>ase</u> <u>case:</u> $F(s_1,\ldots,s_m) \underset{OI}{\Longrightarrow} t_2$. By inspection of
efinition of G, $F(\vec{x})\text{->}t_2 \in P$ where $t_2(\varepsilon)=a \in \Sigma$ and r(
y definition 5.3.1, $p \in \delta(\perp,a)$ and
$\in \delta(p\ [[s_1(\varepsilon),\ldots,s_m(\varepsilon)\ ]],\varepsilon)$ where $m=r(F)$. Hence
$\{(u0,\perp)\} \vee b, t_1) \vdash (\{(u,p)\} \vee b, t_1) \vdash$
$\{(u,F(s_1,\ldots,s_m))\} \vee b, t_1)$.

<u>inductive</u> <u>step:</u> $F(s_1,\ldots,s_m) \overline{\overline{\;\;}}\!\!>_{OI}\; t(s_1,\ldots,s_m) \overline{\overline{\;\;}}_{OI}$ where $n\geq 1$. Depending on the form of $F(\vec{x})\text{->}t\in P$, t are two cases:

<u>case</u> <u>1:</u> $t\in T_{\bar{\Phi}}(X_m)$. By definition 5.3.1, $F\in\delta(t\ [[s_1(\varepsilon),\ldots,s_m(\varepsilon)\ ]],\varepsilon)$, and hence, $(\{(u,\beta(s_1,\ldots,s_m))\}\vee b,t_1) \vdash F(s_1,\ldots,s_m))\}\vee b,t$ By induction, $(\{(uv0,\perp) \mid v\in leaf(t_2)\}\vee b,t_1) \vdash^*$ $(\{(u,t(s_1,\ldots,s_m))\}\vee b,t_1)$.

<u>case</u> <u>2:</u> $t=f(x_1',\ldots,x_q')$ where $f\in\Sigma$, $r(f)=q>0$, and all i, $1\leq i\leq q$, $x_i'\in X_m$. For all i, $1\leq i\leq q$, let $s_i'=s_j$ $x_i'=x_j$. By definition 5.3.1, $p\in\delta(s_1'(\varepsilon),\ldots,s_q'(\varepsilon))$ and $F\in\delta(p(x_1',\ldots,x_q')\ [[s_1(\varepsilon),\ldots,s_m(\varepsilon)\ ]],\varepsilon)$. Henc $(\{(ui,s_i') \mid 1\leq i\leq q\}\vee b,t_1) \vdash (\{(u,p(s_1',\ldots,s_q'))\}\vee$ $\vdash (\{(u,F(s_1,\ldots,s_m))\}\vee b,t_1)$. Clearly, by the definition of $\overline{\overline{\;\;}}\!\!>_{OI}$, for all j, $1\leq j\leq q$, $s_i' \overline{\overline{\;\;}}\!\!>_{OI}^{n_j}$ t where $0<n_j\leq n$. Hence, by induction, for all j, $1\leq$

$(\{(uiv_i0,\perp) \mid j\leq i\leq q,\ v_i\in leaf(t_2/i)\}$
$\vee\{(ui,s_i') \mid 1\leq i<j\}\vee b,t_1) \vdash^*$
$(\{(uiv_i0,\perp) \mid j<i\leq q,\ v_i\in leaf(t_2/i)\}$
$\vee\{(ui,s_i') \mid 1\leq i\leq j\}\vee b,t_1)$.

<u>Lemma</u> **<u>5.3.6:</u>** Given a tree grammar $G=(\bar{\Phi},\Sigma,P,S)$ in Chomsky normal form and its corresponding STPDA $D=(\Sigma,\Gamma,\delta,\perp)$ as defined in definition 5.3.1; any $b\in 2^{\mathbf{N}^*} \times T_\Gamma$; any three trees t_1,t_2,t_3 such that $t_1=t_3[u\text{<-}t_2]$ for some $u\in dom(t_3)$; any $F\in\bar{\Phi}$ where r y sequence of trees $s_1, \ldots, s_m \in T_\Sigma$; and any $n>0$;

(uvO,J_) | v€leaf(t_2)}Vb,t_1) I-^{n}

(u,F(s_1,...,s_m))}Vb,t_1), then F(s_1,...,s_m) $\overset{*}{\Rightarrow}$ t

oof; By induction on n.

se case: id_n - ({(uO,I)}Vb,t..) t-^{2}

(u,F(s_1,...,s_m))}Vb,t_1) where t_2(g)=a€Σ and r(a)=

inspection of definition 5.3.1, clearly F(lt)->a.

nee, F(s_1,...,s_m) $\Rightarrow$ t_2.

ductive step: id_1 I-^{n} id_2 I- id^ where

id_1 = ({(uv0,i) | v€leaf(t_2)}Vb,t_1),

id_3 - ({(u,F(s_1,...,s_m))}Vb,tj), and n≥2. Dependi

the last computation performed, there are two ca

se 1: F€&(t [[s_1(6),...,s_m(6)]],6) where t^I^Cx$_m$

d id_2 - ({(u,t(s_1,...,s_m))}Vb,t). By definition

3.1, it must be the case that F(!?)->t€P. Hence

s_1,...,s_m) $\Rightarrow$ t(s_1,...,s_m). Since t(s_1,...,s_m)4

s_1,...,s_m) $\Rightarrow$ t^ by induction.

se 2: F^(p(x$_1$,...,x$_q$) [[s_1(6),...,s_m(6)]],6) whe

F(*)->f(xj,...,x'$_q$), f€Σ, r(f)=q>0, r(F)=m, and

,...,x'€X$_q$ $_m$. For all i, 1≤i≤q, all j, 1≤j≤m, let

»s^j. if x'$_i$-x$_j$.. By definition 5.3.1,

^((8^(6),...,s^q(6)),f) which is the only way that

computable. That is, id_1 I-$^{n-1}$ id^ \- id_2 ^ i

ere id_4 = ({(ui,s^) | 1≤i≤q}Vb^^ and

id_2 = ({(u,p(s'$_1$,...,s'$_q$))}Vb,tj). From lemma 5.3.3

or each j, $1\leq j\leq q$, $(\{(uiv_i0,\underline{|}) \mid j\leq i\leq q,\ v_i\in leaf(t_2$
$\vee \{(ui,s_i') \mid 1\leq i<j\}\vee b,t_1) \vdash^{n_j}$
$\{(uiv_i0,\underline{|}) \mid j<i\leq q,\ v_i\in leaf(t_2/i)\} \vee$
$(ui,s_i') \mid 1\leq i\leq j\}\vee b,t_1)$ where $0<n_j\leq n-1$. Hence, b
nduction, for all i, $1\leq i\leq q$, $s_i' \overset{}{\underset{OI}{\Longrightarrow}}{}^* t_2/i$. Theref
$(s_1,\ldots,s_m) \underset{OI}{\Longrightarrow} f(s_1',\ldots,s_q') \underset{OI}{\Longrightarrow}{}^* f(t_2/1,s_2',\ldots,s$
$\underset{OI}{\Longrightarrow}{}^* f(t_2/1,t_2/2,s_3',\ldots,s_q') \underset{OI}{\Longrightarrow}{}^* \ldots \underset{OI}{\Longrightarrow}{}^*$
$(t_2/1,t_2/2,\ldots,t_2/q) = t_2$.

heorem 5.3.1: Given any tree grammar $G=(\Phi,\Sigma,P,S)$
eak Chomsky normal form and the corresponding STP
$=(\Sigma,\Gamma,\delta,\underline{|})$ as defined in definition 5.3.1,
$_{OI}(G) = N(D)$.

roof: By the definition of a tree language,
$_{OI}(G) = \{t \mid S \underset{OI}{\Longrightarrow}{}^* t$ where $t\in T_\Sigma\}$. Let $t\in T_\Sigma$ be a
ree such that $S \underset{OI}{\Longrightarrow}{}^* t$. By lemma 5.3.5,
$d_0 = (\{(u0,\underline{|}) \mid u\in leaf(t)\},t) \vdash^* (\{(\varepsilon,S)\},t)$.
efinition 5.3.1, $\underline{|}\in\delta(S,\varepsilon)$, and hence
$\{(\varepsilon,S)\},t) \vdash (\{(\varepsilon,\underline{|})\},t) = id_1$. By definition,
$(D) = \{t\in T_\Sigma \mid id_S = (\{(u0,\underline{|}) \mid u\in leaf(t)\},t)$,
$d_F = (\{(\varepsilon,\underline{|})\},t)$, and $id_S \vdash^* id_F\}$. Clearly id_0 i
nitial configuration and id_1 is a final configura
or D. Hence, $t\in N(D)$ and $L_{OI}(G) \subseteq N(D)$. On the o
and, let $t\in T_\Sigma$ be any tree such that
$d_0 = (\{(u0,\underline{|}) \mid u\in leaf(t)\},t) \vdash^* (\{(\varepsilon,\underline{|})\},t) =$ i
y definition 5.3.1, clearly it must be the case t

$\{(\varepsilon,S)\},t) \vdash id_1$. By lemma 5.3.6, $S \overset{*}{\underset{OI}{\Longrightarrow}} t$.

$_{OI}(G)$ and $N(D) \subseteq L_{OI}(G)$. Therefore

$N(D)$.

nverting STPDAs To TPDAs -

section shows that every STPDA D_1 can be to some TPDA D_2 such that $N(D_1) = N(D_2)$. The in this conversion is to duplicate the symbol at the root of the tree stack for its ding current state.

n 5.3.2: Given a STPDA $D_1=(\Sigma,\Gamma,\delta_1,\perp)$, let the ding TPDA $D_2=(\Gamma,\Sigma,\Gamma,\delta_2,\perp,\perp,\{\perp\})$ such that δ_2 d as follows:

if $F\in\delta_1(\perp,a)$ where $a\in\Sigma$, $F\in\Gamma$, and $r(a)=r(F)=0$, then $(F,F)\in\delta_2(\perp,a)$

if $F\in\delta_1((q_1,\ldots,q_m),f)$ where $f\in\Sigma$, $F\in\Gamma$, $r(f)=r(F)=m>0$, and $q_1,\ldots,q_m\in\Gamma$, then $(F,F)\in\delta_2((q_1,\ldots,q_m),f)$

iii) if $F \in \delta_1(t\,[[F_1,\ldots,F_m]],\varepsilon)$ where $F \in \Gamma$, $t \in T_\Gamma(\mathbf{X}_m)$, $F_1,\ldots,F_m \in \Gamma$, and $t(\varepsilon) \notin \mathbf{X}_m$, $(F,F) \in \delta_2((t(\varepsilon), t\,[[F_1,\ldots,F_m]]),\varepsilon)$

iv) if $F \in \delta_1(t\,[[F_1,\ldots,F_m]],\varepsilon)$ where $F \in \Gamma$, $t \in T_\Gamma(\mathbf{X}_m)$, $F_1,\ldots,F_m \in \Gamma$, and $t(\varepsilon)=x_i$ $1 \leq i \leq m$, then $(F,F) \in \delta_2((F_i, t\,[[F_1,\ldots,F$

Example 5.3.2: Let $D_1=(\Sigma,\Gamma,\delta_1,\underline{|})$ be defined a example 5.2.1. Then, the corresponding TPDA, defined in definition 5.3.2, is $D_2=(\Gamma,\Sigma,\Gamma,\delta_2$ where δ_2 is defined by the following table:

	a	f	g	ε
$\underline{\perp}$	{(a,a)}			
(a)		{(f,f)}		
a,a)			{(g,g)}	
[[a]])				{(F,F)}
[[f]])				{(F,F)}
(f)		{(f,f)}		
f,f)			{(g,g)}	
[[a]])				{(F,F)}
[[f]])				{(F,F)}
[[ ]])				{($\underline{\perp}$,$\underline{\perp}$)}

he following lemma and theorem show that every can be converted to a TPDA. The proofs are and straightforward and have been omitted.

Lemma 5.3.7: Given a STPDA $D_1=(\Sigma,\Gamma,\delta_1,\perp)$ and
corresponding TPDA $D_2=(\Gamma,\Sigma,\Gamma,\delta_2,\perp,\perp,\{\perp\})$ as
definition 5.3.2; any three trees $t_1,t_2,t_3 \in T_{\Sigma}$
that $t_1=t_3[u \leftarrow t_2]$ for some $u \in dom(t_3)$; any $b_1 \in$
any $b_2 \in 2^{\Gamma \times N^* \times T_{\Gamma}}$; any $\beta \in T_{\Gamma}$; and any n>0
$\{(uv0,\perp) \mid v \in leaf(t_2)\} \cup b_1, t_1) \vdash^n (\{u,\beta)\} \cup b_1$
and only if $(\{(\perp,uv0,\perp) \mid v \in leaf(t_2)\} \cup b_2) \vdash^n$
$(\{(\beta(\varepsilon),u,\beta)\} \cup b_2, t_1)$.

Theorem 5.3.2: Given a STPDA $D_1=(\Sigma,\Gamma,\delta_1,\perp)$ an
corresponding TPDA $D_2=(\Gamma,\Sigma,\Gamma,\delta_2,\perp,\perp,\{\perp\})$ as
definition 5.3.2, $N(D_1)=N(D_2)$.

5.3.3 Converting TPDAs To Tree Grammars -

This section shows that every TPDA D can
converted to a tree grammar G such that N(D) =
The idea used in this section resembles the me
to show that a PDA D can be converted to a str
grammar G such that N(D) = L(G) where the nont
of the grammar encode information about the
instantaneous descriptions and the productions
how instantaneous descriptions are updated by
computation relation (see Harrison[78], Lewis
Papadimitriou[81], Schutzenberger[63], and Eve

transform a TPDA into a tree grammar the method
to capture the changes on instantaneous
ions caused by a computation using the set of
nals. To accomplish this, a nonterminal
of three components. For any computation
d_2 where URH($id_1 \vdash id_2$)=(q,u,p), the state q
current stack root p(ϵ) of the updated
d (q,u,p) at tree address u are encoded in the
nding nonterminal to define the first two
ts. The third component of the nonterminal is
look-back references which is a tuple
ng previous instantaneous descriptions that
e existed in order for the current
neous description to exist. In particular, the
instantaneous descriptions encoded are the
ociated with the immediate descendants of the
node u. Furthermore, the third component is a
pairs consisting of the state and root of the
ck associated with each of the immediate
nts of that node (i.e. the same idea as the
o components of the nonterminal except they
e first two components of each of its immediate
nts).

Another way of viewing this transformation is
e nonterminals trace the history of how an
stantaneous description is reached. Loosely
eaking, a nonterminal $(q,F,((q_1,F_1),\ldots,(q_m,F_m)))$
ates the following: Beforehand, there were m
ad-heads where each read-head i, $1\leq i\leq m$, had state
d tree stack s_i associated with it such that the
tree stack s_i was labeled with the stack symbol
ter several computations, the m read-heads have b
rged together where the current state associated
e merged read-heads is the state q and the associ
ee stack is $F(s_1,\ldots,s_m)$.

To assist in the conversion of the TPDA into a
ee grammar, the definition below defines the func
e" where "le" takes a tree t and returns a set of
ees (labeled by nonterminals) where each tree is
story of a computation which might have produced
ee stack $t(s_1,\ldots,s_m)$. Furthermore, each tree st
can be arbitrarily chosen since the function "le
ly uses the root of the tree stack s_i.

<u>**finition**</u> <u>**5.3.3:**</u> Given a TPDA $D=(K,\Sigma,\Gamma,\delta,q_0,\perp,Q)$,

$$: K \times T_{\Gamma}(\mathbf{X}_m) \times \text{tuple}_m(K \times \Gamma)$$

$$\rightarrow 2^{T(K \times \Gamma \times \text{tuples}(K \times \Gamma))(\mathbf{X}_m)}$$

the lookahead expansion function recursively defi
h that

$le(q,t,((q_1,F_1),...,(q_m,F_m))) = t(\epsilon)$ if $t(\epsilon) \in X_m$,

$le(q,t,((q_1,F_1),...,(q_m,F_m))) =$

$\{(q,t(\epsilon),((p_1,G_1),...,(p_n,G_n)))(t_1,...,t_n) \mid$

$r(t(\epsilon))=n$, $((p_1,G_1),...,(p_n,G_n)) \in tuple_n(K \times T$

for all i, $1 \leq i \leq n$, if $t(i) \in X_m$, then

$(p_i,G_i)=(q_j,F_j)$ where $t(i)=x_j$, otherwise $G_i=t$

and $t_i \in le(p_i,t/i,((q_1,F_1),...,(q_m,F_m)))\}$

otherwise.

In other words, the function "le" is used to
cribe a long sequence of computation moves. The
putation being described started with an
tantaneous description where p of its read-heads
state q_i and the tree stack s_i associated with i
some i, $1 \leq i \leq j$ such that the root of tree stack
F_i, and after a long sequence of computation move
p read-heads were merged into a single read-head
re the resulting associated state and tree stack
nd $t(s_1,...,s_m)$ respectively. Hence, given that
ulting instantaneous description has a read-head
ated at tree address u, the function takes the

following three arguments: The state q associ

the read-head at u, the tree t where $t(s_1,\ldots,$

the tree stack associated with the read-head a

the tuple $((q_1,F_1),\ldots,(q_m,F_m))$ where for all

$s_i(\varepsilon)=F_i$ and the state associated with the tre

s_i is q_i. Using the arguments, the function l

a set of trees labeled with nonterminals where

tree traces a possible history of how the tree

$t(s_1,\ldots,s_m)$ was generated from the original s

tree stacks. In other words, it guesses the

intermediate states that the TPDA must have go

through in order to produce the tree stack t(s

from the sequence of tree stacks $s_1,\ldots,s_m$.

Example: Let $D=(K,\Sigma,\Gamma,\delta,q_0,Q)$ be a TPDA where

and $a,f,F,g\in\Gamma$ where $r(a)=0$, $r(f)=2$, $r(F)=1$, a

Then,

```
le(1,a,((2,f))) = {(1,a,∅)},

le(1,  g  ,((2,a),(1,f))) = {(1,  g  , ((2,a),(
      / \                         / \
     x   y                       x   y

le(1,  g  ,((2,a),(1,f))) = {(1,  g  , ((1,f),(
      / \                         / \
     y   x                       y   x
```

```
d le(1, f ,((2,a))) =
       / \
      x   F
          |
          f
          |
          x

        { (1,f,((2,a),(1,F))) ,
             /         \
            x     (1,F,((1,f)))
                         |
                  (1,f,((2,a)))
                         |
                         x

          (1,f,((2,a),(1,F))) ,
             /         \
            x     (1,F,((2,f)))
                         |
                  (2,f,((2,a)))
                         |
                         x

          (1,f,((2,a),(2,F))) ,
             /         \
            x     (2,F,((1,f)))
                         |
                  (1,f,((2,a)))
                         |
                         x

          (1,f,((2,a),(2,F))) }
             /         \
            x     (2,F,((2,f)))
                         |
                  (2,f,((2,a)))
                         |
                         x
```

ition 5.3.4: Given a TPDA $D=(K,\Sigma,\Gamma,\delta,q_0,\underline{|},Q)$, le

orresponding tree grammar $G=(\Phi,\Sigma,P,B)$ be defined

:

$\Phi = \{(q,F,((q_1,F_1),\ldots,(q_m,F_m))) \mid$

$q \in K$, $F \in \Gamma$, $r(F)=m$, and

$((q_1,F_1),\ldots,(q_m,F_m)) \in tuple_m(K \times \Gamma)\} \vee$

$r((q,F,((q_1,F_1),\ldots,(q_m,F_m))))=m$ and r(B

and P is constructed as follows:

i) $B \rightarrow (q,\perp,\emptyset)$ for all $q \in Q$

ii) if $(q,F) \in \delta(q_0,a)$, then $(q,F,\emptyset) \rightarrow a \in$ P

iii) if $(q,F) \in \delta((q_1,\ldots,q_m),f)$ then for ever

sequence of stack symbols $F_1,\ldots,F_m \in \Gamma$,

$(q,F,((q_1,F_1),\ldots,(q_m,F_m)))(\vec{x}) \rightarrow f(\vec{x}) \in$

iv) if $(q_2,F) \in \delta((q_1,t\,[[F_1,\ldots,F_m]]),\varepsilon)$, the

every sequence of states $p_1,\ldots,p_m \in K$, e

$t' \in le(q_1,t,((p_1,F_1),\ldots,(p_m,F_m)))$,

$(q_2,F,((p_1,F_1),\ldots,(p_m,F_m)))(\vec{x}) \rightarrow t' \in$ P

Note: Productions built by condition (i) essen
state that the goal is to go from an initial
instantaneous description to a final instantan
description. Productions built by condition (
that if the updated read-head was produced by
leaf, then the previous instantaneous descript
the current instantaneous description was prod
was an initial description and terminates the

putation. Productions built from condition tate that if a read-move was performed on an l node, the resulting updated read-head came instantaneous description in a single move and ated read-head should be broken down into the tions of the m read-heads that were merged into the read-move. Finally, productions built from on (iv) state that if the updated read-head was d by a reduce-move, the sequence of computations by the pop on the tree stack must be guessed, produces a production for each possible guess.

mma 5.3.9, lemma 5.3.10, and theorem 5.3.3 show that for any TPDA D and the corresponding ammar G defined by definition 5.3.4, $L_{OI}(G)$. Lemma 5.3.9 shows that for any tion $id_1 \vdash^n id_2$ where id_1 is an initial aneous description on the tree t and the last tion produces the updated read-head $s_1,\ldots,s_m)$), there exists a tree $,F(s_1,\ldots,s_m),\emptyset)$ such that $t' \overset{*}{\underset{OI}{\Longrightarrow}} t/u$. Lemma shows the converse of lemma 5.3.9. It shows r any tree $t' \in le(q,t'',\emptyset)$, if $t' \overset{n}{\underset{OI}{\Longrightarrow}} t \in T_\Sigma$, then tree $s \in T_\Sigma$ such that s=s[u<-t], $id_1 \vdash^* id_2$ where the initial instantaneous description and the dated read-head of the last computation is (q,u,t

nally, theorem 5.3.3 uses the results of these tw

mmas to show the desired result that $N(D)=L_{OI}(G)$.

mma 5.3.9: Given a TPDA $D=(K,\Sigma,\Gamma,\delta,q_0,\perp,Q)$ and t

rresponding tree grammar $G=(\Phi,\Sigma,P,B)$ as defined i

finition 5.3.3; any $b\in 2^{K \times N^* \times T_\Gamma}$; any three

$,t_2,t_3\in T_\Sigma$ such that $t_1=t_3[u\leftarrow t_2]$ for some $u\in dom(t$

y sequence of trees $s_1,\ldots,s_m\in T_\Gamma$; any $n>0$; if

$(q_0,u0,\perp)$ | $u\in leaf(t_2)\}\vee b,t_1) \vdash^n$

$(q,u,F(s_1,\ldots s_m))\}\vee b,t_1)$, then there exists a tr

$\in le(q,F(s_1,\ldots,s_m),\emptyset)$ such that $t' \overset{*}{\underset{OI}{\Longrightarrow}} t_2$.

oof: By induction.

se case: $(\{(q_0,u0,\perp)$ | $u\in leaf(t_2)\}\vee b,t_1) \vdash$

$(q,u,F)\}\vee b,t_1)$ where $t_2(\varepsilon)=a\in\Sigma$ and $r(a)=0$. By

finition 5.3.3, $(q,F,\emptyset)\to a\in P$. Hence $(q,F,\emptyset) \underset{OI}{\Longrightarrow}$

ductive step: $id_1 \vdash^n id_2 \vdash id_3$ where

$_1=(\{(q_0,u0,\perp)$ | $u\in leaf(t_2)\}\vee b,t_1)$ and

$_3=(\{(q,u,F(s_1,\ldots,s_m))\}\vee b,t_1)$. Depending on the

st computation performed, there are two cases:

se 1: $id_2=(\{(q_i,ui,s_i)$ | $1\leq i\leq m\}\vee b,t)$ where

$,F)\in\delta((q_1,\ldots,q_m),f)$ and $r(F)=m$. By definition o

, clearly for each j, $1\leq j\leq m$,

$(q_0,uiv_i0,\perp)$ | $j\leq i\leq m$, $V_i\in leaf(t_2/i)\}\ \vee$

(q_i,ui,s_i) | $1\leq i<j\}\vee b,t_1) \vdash^{n_j}$

$(q_0, uiv_i 0, \perp) \mid j<i\leq m,\ v_i \in leaf(t_2/i)\}$

$\{(q_i, ui, \perp) \mid 1\leq i\leq j\} \vee b, t_1)$ where $0<n_j\leq n$. By
duction, for all i, $1\leq i\leq m$, there exists trees
$\in le(q_i, s_i, \emptyset)$ such that $t_i' \overset{*}{\underset{OI}{\Longrightarrow}} t_2/i$. By definiti
3.3,

$, F, ((q_1, s_1(\varepsilon)), \ldots, (q_m, s_m(\varepsilon)))(\vec{x}) \rightarrow f(x_1, \ldots, x_m) \in$
nce, $(q, F, ((q_1, s_1(\varepsilon)), \ldots, (q_m, s_m(\varepsilon)))(t_1', \ldots, t_m')$
$t_1', \ldots, t_m') \underset{OI}{\Longrightarrow} f(t_2/1, t_2', t_3', \ldots, t_m') \underset{OI}{\Longrightarrow} \ldots \underset{OI}{\Longrightarrow}$
$t_2/1, t_2/2, \ldots, t_2/m) = t_2$.

se 2: $id_2 = (\{(q', u, t(s_1, \ldots, s_m))\} \vee b, t_1)$ where
$, F) \in \delta((q', t\,[[s_1(\varepsilon), \ldots, s_m(\varepsilon)\,]]), \varepsilon)$. By inductio
ere exists a set of states $p_1, \ldots, p_m \in K$ such that

i) $t'' \in le(q', t, ((p_1, s_1(\varepsilon)), \ldots, (p_m, s_m(\varepsilon))))$

ii) for each i, $1\leq i\leq m$, $s_i' \in le(p_i, s_i, \emptyset)$

iii) $t''(s_1', \ldots, s_m') \underset{OI}{\Longrightarrow} t_2$.

definition 5.3.3,

$, F, ((p_1, s_1(\varepsilon)), \ldots, (p_m, s_m(\varepsilon))))(\vec{x}) \rightarrow t'' \in P$. Hence
$, F, ((p_1, s_1(\varepsilon)), \ldots, (p_m, s_m(\varepsilon))))(s_1', \ldots, s_m') \underset{OI}{\Longrightarrow}$
$(s_1', \ldots, s_m') \overset{*}{\underset{OI}{\Longrightarrow}} t_2$.

mma 5.3.10: Given any TPDA $D=(K, \Sigma, \Gamma, \delta, q_0, \perp, Q)$ an
rresponding tree grammar $G=(\Phi, \Sigma, P, B)$ as defined
finition 5.3.3; any $b \in 2^{K \times \mathbf{N}^* \times T_\Gamma}$; any three
$, t_2, t_3 \in T_\Sigma$ such that $t_1 = t_3[u \leftarrow t_2]$ for some $u \in dom($

any tree $t \in T_{\Gamma}$; any $q \in K$; any tree $t' \in le(q,t,$

$n>0$; if $t' \overset{}{\underset{OI}{\Rightarrow}}{}^{n}\ t_2$, then

$(\{(q_0,uv0,\perp) \mid v \in leaf(t_2)\} \vee b, t_1) \vdash^*$

$(\{(q,u,t)\} \vee b, t_1)$.

Proof: By induction on n.

base case: $(q,F,\emptyset) \underset{OI}{\Rightarrow} a$ where $(q,F) \in \delta(q_0,a)$

$(q,F,\emptyset) \rightarrow a \in P$. Clearly, $(\{(q_0,u0,\perp)\} \vee b, t_1) \vdash$

$(\{(q,u,F) \vee b, t_1)$.

inductive step: $t' \underset{OI}{\Rightarrow}{}^{n+1}\ t_2$. depending on t

derivation step, there are two cases (either a

production produced from a shift-move on an i

node, or a production produced from a reduce-m

case 1: $(q,F,((q_1,s_1(\varepsilon)),\ldots,(q_m,s_m(\varepsilon))))(s'_1$

$\underset{OI}{\Rightarrow} f(s'_1,\ldots,s'_m) \underset{OI}{\Rightarrow}{}^{n}\ t_2$ where

i) $(q,F,((q_1,s_1(\varepsilon)),\ldots,(q_m,s_m(\varepsilon))))(\vec{x})$

$\rightarrow f(x_1,\ldots,x_m)$

ii) $(q,F) \in \delta((q_1,\ldots,q_m),f)$

iii) $s'_i \in le(q_i,t/i,\emptyset)$ for all i, $1 \leq i \leq m$.

By the definition of $\underset{OI}{\Rightarrow}$, clearly, for all i,

$s'_i \underset{OI}{\Rightarrow}{}^{n_i}\ t_2/i$ where $0<n_i \leq n$. Hence, by inducti

each j, $1 \leq j \leq m$, $(\{(q_0,uiv_i0,\perp) \mid j \leq i \leq m,\ v_i \in leaf$

$\vee\ (\{(q_i,ui,t/i) \mid 1 \leq i < j\} \vee b, t_1) \vdash^*$

$(\{(q_0,uiv_i0,\perp) \mid j<i \leq m,\ v_i \in leaf(t_2/i)\} \vee$

[$_1$,ui,t/i) I Ki<j}Vb,t_1). Also, since
,F)€δ((q_1,...,q_m),f), clearly
(q_1,ui,t/i) | Ki<m}Vb,t_1) ⊢
;q,u,F(t/l,...,t/m))}Vb,t_1) where F(t/1,...,t/m)=

se 2: (q_2,F,((p_1,s_1(€)),...,(p_m,s_m(€))))(s'_1,...,
> t'(s[. . . . .s^) $\overset{}{\underset{\delta I}{==>}}^n$ t_2 where

i) t′€le(q_1,j3,((p_1,s_1(6)),...,(p_m,s_m)))

ii) for each i, l<i<m, s^€le(p_{i}>t/i,0)

ill) ($q_{;J!}$,F)€δ((q_1,p [[s_l(6),...,s_m<6)]]),£).

induction, ({(q_Q,uvO,J_) | v€leaf(t_2)}Vb,tj) k *
[q_1 ,u,3(t/1,...,t/m))}Vb,t_1). From above, clear]
(q_1,u,β(t/1,...,t/m))}Vb,t_1) ⊢
[q_2,u,F(t/l,...,t/m))}Vb,t_1) where F(t/1,...,t/m}

iorem 5.3.3: Given any TPDA D=(K,Ī, p ,q_Q ,J_,Q) and
rresponding tree grammar G=($,i.,P ,B) as defined i
Einition 5.3.3, L_{QI}(G) - N(D).

aof; By the definition of L_{nT}(G),
$_E$(G) » {t€Ty I B $\underset{\delta I}{=>}$* t}. By the definition of N(
D) = {t€T^ | id_s = ({(q_o,uO,i) | u€leaf(t)},t),
p - ({(q,6,i)},t), q€Q, and id_g I- * id_p}. Let t€
any tree such that B QY> t. By the definition o
>, and definition 5.3.3, there must exist a q€Q s

that $B \underset{OI}{\Longrightarrow} (q,\perp,\emptyset) \underset{OI}{\Longrightarrow}^{*} t$. Clearly $\{(q,\perp,\emptyset)\} = le(q,\perp,\emptyset)$. Hence, by lemma 5.3.11 $(\{(q_0,u,\perp) \mid u \in leaf(t)\},t) \vdash^{*} (\{(q,\varepsilon,\perp)\},t)$. then, $t \in N(D)$ and hence $L_{OI}(G) \subseteq N(D)$. On the hand, let $t \in T_{\Sigma}$ be any tree such that $t \in N(D)$. definition, $(\{(q_0,u0,\perp) \mid u \in leaf(t)\},t) \vdash^{*}$ $(\{(q,\varepsilon,\perp)\},t)$ for some $q \in Q$. By lemma 5.3.10, exists a tree $t' \in le(q,\perp,\emptyset)$ such that $t' \underset{OI}{\Longrightarrow}^{*} t$ definition of le, $t'=(q,\perp,\emptyset)$. By definition 5 $B \underset{OI}{\Longrightarrow} (q,\perp,\emptyset)$. Hence $t \in L_{OI}(G)$ and $N(D) \subseteq L_{OI}($ Therefore $L_{OI}(G)=N(D)$.

5.3.4 Comparing Classes Of Tree Languages -

This section uses the three previous sect theorem 4.9.1 from chapter four, to show that of tree grammars, TPDAs, and STPDAs, are ident

For convenience of notation, let the clas languages generated by tree grammars under an derivation, the class of tree languages genera tree grammars in weak Chomsky normal form unde derivation, the class of tree languages accept TPDAs, and the class of tree languages accepte STPDAs be denoted as C_{TG}, C_{WCNF}, C_{TPDA}, and C_S

ctively.

em **5.3.4:** $C_{TG} = C_{WCNF} = C_{TPDA} = C_{STPDA}$.

: By theorem 4.9.1, $C_{TG} \subseteq C_{WCNF}$. By theorem

, $C_{WCNF} \subseteq C_{STPDA}$. By theorem 5.3.2,

$A \subseteq C_{TPDA}$. By theorem 5.3.3, $C_{TPDA} \subseteq C_{TG}$. Hence

$C_{WCNF} = C_{STPDA} = C_{TPDA}$.

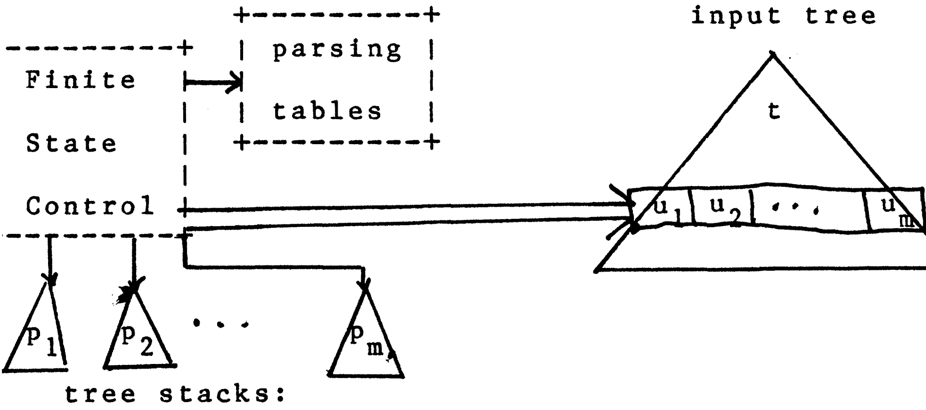
Chapter VI

THE BUTLR(0) PARSER

This chapter presents a new construction m build deterministic bottom-up tree parsers for subclass of the context-free tree languages. T of parser presented is a BUTLR(0) parser (a tre which parses trees bottom-up by lifting LR(0) p techniques for strings). Logically, the contro BUTLR(0) parser can be viewed as consisting of parts, a driver routine and three parsing table figure 6.1.1). In constructing the BUTLR(0) pa the parsing tables are dependent on the given t grammar, and must be constructed, while the dri routine remains the same for all tree grammars.

```
                              +-----------+
                              |   table   |     parsing
         tree grammar -->|           |-->
                              | generator |     tables
                              +-----------+
```

a) generating the parsing tables



b) operation of BUTLR parser

Figure 6.1.1 : Layout of BUTLR parser

A BUTLR(0) parser is a different presentation of
STPDA. The transition function δ of the BUTLR(0)
er is implicitly defined by a set of parsing tabl
rated from the given tree grammar. Furthermore,
e parsing tables can be viewed as a "compressed"
esentation of the transition map δ.

In generating the BUTLR(0) parser, from a
tree grammar, the tables are built to simulate
derivation in reverse. Hence, the object of th
construction method is to attempt to maintain t
property that every tree stack corresponds to a
of some legal sentential form. This is done by
building a bottom-up tree automaton (called the
characteristic automaton) which parses each tre
to recognize which sentential form each tree st
could be a subtree of. Like an LR(0) parser, o
BUTLR(0) characteristic automaton is built, the
BUTLR(0) parser can be constructed directly fro
characteristic automaton.

This chapter begins by presenting the BUTL
parser in terms of its parsing tables and an ex
a BUTLR(0) parser. The chapter continues in se
6.2 by presenting "characteristic trees" and th
corresponding characteristic automaton to parse
trees. The chapter concludes with section 6.3
presents the algorithm to convert the BUTLR(0)
characteristic automaton into a BUTLR(0) parser
the correctness of the BUTLR(0) parser construc
method, and presents some conjectures as to whe
construction method will produce a deterministi
BUTLR(0) parser.

BUTLR(0) Parsing Tables

A BUTLR(0) parser is a machine which has a tre
ut, uses tree stacks as internal memory, and use
ee parsing tables to define the transition funct
e figure 6.1.1). More formally, a BUTLR(0) pars
a sextuple M=(G, K, shift, reduce, goto, start)
re:

G = (Φ,Σ,P,S) is the tree grammar defining the BUTLR(0) parser;

K is a finite ranked alphabet of parser <u>states</u>;

<u>shift</u> : tuples(K) x Σ -> K$\vee${<u>error</u>} is a functio defining the parsing <u>shift</u> <u>table</u>;

<u>reduce</u> : K -> 2^P is a function defining the parsing <u>reduce</u> <u>table</u>;

<u>goto</u> : tuples(K) x Φ -> K$\vee${<u>error</u>} is a function defining the parsing <u>goto</u> <u>table</u>; and

<u>start</u>$\in$K is the <u>initial</u> <u>state</u> and denotes the <u>empty</u> <u>tree</u> <u>stack</u>.

As mentioned above, a BUTLR(0) parser is just
fferent presentation of a STPDA. Hence, an
stantaneous description of a BUTLR(0) parser (den
) is the same as for a STPDA. An instantaneous
scription consists of a pair (a,t)$\in 2^{\mathbf{N}^*} \times T_K \times T_\Sigma$
ere t is the input tree and a is a set of pairs (

ere u is the tree address of a node covered by a
ad-head and p is the corresponding tree stack
sociated with that read-head. Likewise, the init
nfiguration of a BUTLR(0) parser is the instantan
scription ({(u0,<u>start</u>) | u$\in$leaf(t)},t) where t is
ee to parse.

The decision relation $\vdash_d \subseteq$ SID x SID of a BUTL
rser M=(G=($\bar{\Phi}$,Σ,P,S), K, <u>shift</u>, <u>reduce</u>, <u>goto</u>, <u>star</u>
the computation relation for the BUTLR(0) parser
d determines the next move of the BUTLR(0) parser
wever, before describing this relation, let me
troduce a help function
eleton : $T_{\Sigma \vee \bar{\Phi}}(X_m) \to 2^{T_K(X_m)}$ which generates a set
ees where each tree can be viewed as a possible
ld-card match to a production's right-hand side.
ven any tree t$\in T_{\Sigma \vee \bar{\Phi}}(X_m)$,

skeleton(t) = {s$\in T_K(X_m)$ | dom(t)=dom(s),
t(u)=s(u) if u$\in$var(t), and
r(t(u))=r(s(u)) for all u$\in$dom(t)}

<u>Note:</u> This function is used to perform the corresponding operation on a STPDA, as the opera of popping n symbols from the stack in a LR(0) parser when reducing on a production A->α where length(α)=n.

[ample 6.1.1: Let G^C 3),i,P,S) be a tree grammar si

Lat

3> - {S,F} where r(S)=O and r(F)=*l;

1 - {a,f,g} where r(a)=O, r(f)-l, and r(g)=2; «

```
P = {S -> F , F -> F , F ->  g  }•
          II       II        / \
          ax     f    x    x    x
                   I
                   x
```

irthermore, assume that the set of parser states 1

:fined such that K={1,2,3,4,5,6} where r(l)=r(2)=i

;3)=r(4)=r(6)=l, and r(5)=2. Then, by the above

ifinition,

```
skeleton(F)   - { 3 , 3 , 4 , 4 , 6 , 6 } ,
         I      I   I   I   I   I   I
         a       1   2    1   2    1   2

skeleton(F)   = { 3 , 3 , 3 , 4 , 4 , 4 , 5 , 5 ,
         I      I   I   I   I   I   I   I   I
         f          3   4   6   3   4   6   3   4
         1      I   I   I   I   I   I   I   I
         x          x   x   x   x   x   x   x   x

and skeleton(  g  ) • {  5  }.
              / \        / \
              xx          xx
```

Having defined the function "skeleton", the

jcision relation is defined as follows: Given an

istantaneous descriptions id_1 and id^, id_1 K id«

nd only if one of the following conditions hold

i) $id_1 = (\{(u0,\underline{start}\}\vee b,t)$ and

$id_2 = (\{(u,q)\}\vee b,t)$ where $b\in 2^{\mathbf{N}^*} \times T_K$

$u\in dom(t)$, $t(u)=a\in\overline{\Sigma}$, $r(a)=r(q)=0$, and

$\underline{shift}(\underline{start},a)=q$.

ii) $id_1 = (\{(ui,p_i) \mid 1\leq i\leq m\}\vee b,t)$ and

$id_2 = (\{(u,q(p_1,\ldots,p_m))\}\vee b,t)$ where

$b\in 2^{\mathbf{N}^*} \times T_K$, $u\in dom(t)$, $t(u)=f\in\overline{\Sigma}$, $q\in K$,

$r(f)=r(q)=m>0$, for all i, $1\leq i\leq m$, ui∈d

$p_i(\varepsilon)=q_i$, and $q=\underline{shift}((q_1,\ldots,q_m),f)$

iii) $id_1 = (\{(u,\beta(t_1,\ldots,t_m)\}\vee b,t)$ and

$id_2 = (\{(u,q(t_1,\ldots,t_m))\}\vee b,t)$ where

$b\in 2^{\mathbf{N}^*} \times T_K$, $q\in K$, $F\in\Phi$, $F\neq S$, $r(q)=r(F)=$

sequence of trees $t_1,\ldots,t_m\in T_K$ such t

all i, $1\leq i\leq m$, $t_i(\varepsilon)=q_i$,

$F(\vec{x})\to s\in\underline{reduce}(\beta(t_1,\ldots,t_m)(\varepsilon))$,

$\beta\in skeleton(s)$, and if $r(F)=0$, then

$\underline{goto}(\underline{start},F)=q$, otherwise

$\underline{goto}((q_1,\ldots,q_m),F)=q$

iv) $id_1 = (\{(\varepsilon,\beta),t)$ and $id_2 = (\{(\varepsilon,\underline{start}$

where $S\to s\in\underline{reduce}(\beta(\varepsilon))$, and $\beta\in skelet$

In other words, condition (i) is a shift-move

leaf of the input tree, condition (ii) is a sh

over an internal node, condition (iii) is a re

l on the production $F(\vec{x})$->s, and condition (iv) i
luce-move on the start production S -> s. Note
condition (iii) is not as complicated as it look
ever the function reduce, defined by the state
lling the root of the tree stack, has the
action $F(\vec{x})$->s as one of its elements, and trees
igh t_m can be found to match occurrences of its
ables where only the skeleton of s must match the
β and all occurrances of tree t_i corresponding t
tions of x_i are identical, then a reduction can b
ormed. Should a reduction be performed, the symb
used to reunite the trees t_1 through t_m and is
rmined by using the goto function on the roots of
trees $t_1,\ldots,t_m$ and the nonterminal F.

By the above definition, one should also note th
difference between the computation relation $\vdash$ fo
PDA, and the computation relation $\vdash_d$ for a BUTLR(
er is that the reduce-move defined by an entry in
reduce parsing table and the goto parsing table
ne a set of possible computations using $\vdash$. Henc
reduce and goto tables are a compressed
esentation of a set of reduce-moves in a STPDA.

A BUTLR(0) parser is considered <u>well</u> <u>defined</u>
nd only if the BUTLR(0) parser is deterministic (
s conservative and does not contain any shift/red
r reduce/reduce conflicts). In other words, a
UTLR(0) parser is well defined if and only if

i) G is conservative

ii) for all $k \in K$, $|\underline{reduce}(k)| \underline{<} 1$

iii) for all $f \in \Sigma$ where $r(f)=n$, for all tuples
$(k_1,\ldots,k_n) \in tuples(K)$, if
$\underline{shift}((k_1,\ldots,k_n),f) \in K$, then for all i, 1
$\underline{reduce}(k_i)=\emptyset$.

iv) For all $a \in \Sigma$ where $r(a)=0$, for all states
if $\underline{shift}(k,a) \in K$, then $\underline{reduce}(k)=\emptyset$.

ote that condition (i) protects against infinite
ondeterminism embedded in the definition of a
educe-move, condition (ii) guarantees that there
ot be any reduce/reduce conflicts, conditions (ii
uarantees that there will not be shift/reduce con
n a terminal symbol with rank>0, and condition (i
uarantees that there will not be a shift/reduce
onflict with a constant terminal symbol.

Acceptance of an input tree t occurs if and or the BUTLR(0) parser can reach the root of the ir ee and have an empty tree stack. More formally, ee language accepted by a BUTLR(0) parser M, denc M), is the set

$$N(M) = \{t \in T_{\Sigma} \mid (\{(u0,\underline{start}) \mid u \in leaf(t)\},t) \vdash_d^{*} (\{(\varepsilon,\underline{start})\},t)\}$$

ere $\vdash_d^{*}$ is the transitive reflexive closure of $\vdash_d$

ample 6.1.2: Let $G=(\Phi,\Sigma,P,S)$ be the tree grammar fined in example 6.1.1. The language generated l trees of the form

```
              g
            /   \
       /-- f     f --\
      |    |     |    |
      |    .     .    |
   n <     .     .     > n        where n>0
      |    .     .    |
      |    |     |    |
       \-- f     f --/
           |     |
           a     a
```

ie BUTLR(0) parser M to recognize G is the tuple (G, K, shift, reduce, goto, 1) such that

K = {1,2,3,4,5,6} where r(1)=r(2)=0,

r(3)=r(4)=r(6)=1, and r(5)=2; and

shift, reduce, and goto are defined by the foll tables where blank (or omitted) entries represer error values:

```
                 shift                 reduce

             a    f    g
          +----+----+----+       +---------+
     1    | 2  |    |    |     3 |{ S->F }|    (2
          +----+----+----+       |     |   |
   (2)    |    | 4  |    |       |     a   |    (4
          +----+----+----+       +---------+
 (2,2)    |    |    | 5  |     5 |{F-> g }|
          +----+----+----+       | |   / \ |
 (2,4)    |    |    | 5  |       | x x    x|
          +----+----+----+       +---------+
 (4,2)    |    |    | 5  |     6 |{ F->F }|
          +----+----+----+       |   |  |  |
 (4,4)    |    |    | 5  |       |   x  f  |
          +----+----+----+       |      |  |
   (4)    |    | 4  |    |       |      x  |
          +----+----+----+       +---------+
```

```
For example, the tree t = g   is accepted
                         / \
                        f   f
                        |   |
                        f   f
                        |   |
                        a   a

({(1110, 1), (2110, 1)}, t) ⊢d

({(111, 2), (2110, 1)}, t) ⊢d

({(11, 4), (2110, 1)}, t) ⊢d
       |
       2

({(1, 4), (2110,1)}, t) ⊢d
      |
      4
      |
      2

({(1, 4), (211, 2)}, t) ⊢d
      |
      4
      |
      2
```

```
l, 4), (21, 4)}, t) ⊢d
   |        |
   4        2
   |
   2

l, 4), (2, 4)}, t) ⊢d
   |       |
   4       4
   |       |
   2       2

ε,  5  )}, t) ⊢d
   / \
  4   4
  |   |
  4   4
  |   |
  2   2

ε, 6)}, t) ⊢d
   |
   4
   |
   4
   |
   2

ε, 6)}, t) ⊢d
   |
   4
   |
   2

ε, 3)}, t) ⊢d
   |
   2

ε, 1)}, t) which is the accepting condition.

ilarly, the tree t = g  is rejected as follows:
                    / \
                   f   a
                   |
                   a
```

({(110, 1), (20, 1)}, t) $\vdash_d$

({(11, 2), (20, 1)}, t) ^

({(1, 4), (20, 1)}, t) $\vdash_d$
      |
      2

({(1, 4), (2, 2)}, t)
      |
      2

The parse fails at this point since there are
legal moves, and none of the above instantaneo
descriptions is an accepting condition. Furth
the BUTLR(O) parser is well defined (i.e. is
deterministic)•

6.2 The BUTLR(O) Characteristic Automaton

As stated earlier, a BUTLR(O) parser M is
generated using a construction method which li
techniques used in LR(0) parsers. Therefore,
to find a way to generate M, the new construct
method should try to maintain the property tha
input tree t, if the subtree t/u is scanned by
BUTLR(O) parser, and its corresponding tree st
represents some tree s^T-r-w-«, then the followi
conditions should hold:

if $t \in L(G)$, then $S \overset{*}{\underset{OI}{\Longrightarrow}} t''[u\text{<-}s] \overset{*}{\underset{OI}{\Longrightarrow}} t$

there exists a tree $t' \in T_{\Sigma}$ such that $S \overset{*}{\underset{OI}{\Longrightarrow}}$ $t''[u\text{<-}s] \overset{*}{\underset{OI}{\Longrightarrow}} t'$.

r words, the construction method should maintain perty that every tree stack, in an instantaneous tion, corresponds to the subtree of some legal ial form. Condition (i) states that this will case whenever the input is legal while condition ates that even if the input tree is illegal, till exists some tree t′ such that the tree is a legal subtree of the corresponding legal ial form.

other way of viewing the above condition is that struction method should produce a BUTLR(0) where every reduce-move will be defined to the inverse of some OI derivation step, and the of every possible OI derivation step is defined reduce-move. Hence, for any sentential form $t_1,\ldots,t_m)]$ and any production $F(\vec{x})\text{->}s \in P$, if the instantaneous description contains the tree $(t_1,\ldots,t_m)$, the construction method should a reduce-move such that the tree stack $.,t_m)$ will be updated to $F(t_1,\ldots,t_m)$. To ish this, one must have a way of recognizing all possible stack configurations in which a reduce

should be defined (i.e. when the reverse of an

derivation step should be performed). The set

stack configurations for which a reduce-move sh

defined is the set $\{s(t_1,\ldots,t_m) \mid S \overset{*}{\underset{OI}{\Longrightarrow}}$ $t[u \leftarrow F(t_1,\ldots,t_m)] \underset{OI}{\Longrightarrow} t[u \leftarrow s(t_1,\ldots,t_m)]\}$. Ea

$s(t_1,\ldots,t_m)$ in the set is called a <u>characteris</u>

and the set will be denoted as CT_G.

If one is able to lift LR(0) parsing techn

verbatim, one would expect the set CT_G to be re

However, this is not the case as shown by the f

theorem:

<u>Theorem 6.2.1:</u> The class of languages defined b

characteristic trees of tree grammars is not co

in the class of regular tree languages.

<u>Proof:</u> Let $G=(\Phi,\Sigma,P,S)$ be a tree grammar s.t.

$\Phi = \{S,F\}$ where $r(S)=0$ and $r(F)=1$;

$\Sigma = \{a,f,g\}$ where $r(a)=0$, $r(f)=1$, and $r(g)=2$

```
P = {S->F  ,  F->F  ,  F-> g  }.
        |       |   |     |  / \
        a       x   f     x x   x
                    |
                    x
```

By definition, any tree $t \in CT_G$ is in one of the

following two forms:

```
(i)        F
           |
           f
           |
           .
           .      n        where n>0
           .
           |
           f
           |
           a

(ii)       g
        /     \
     f           f
     |           |
     .           .
n    .           .    n        where n>0.
     .           .
     |           |
     f           f
     |           |
     a           a
```

Clearly, by theorem 4.10.1 which presents the ping lemma for regular tree languages, there exis ufficiently large n such that trees of the second n can not be regular. Hence CT_G can not be regul

Having failed to lift up to characteristic tree corresponding fact that characteristic strings a ular in LR(0) parsers, a natural question to ask t class of tree languages the characteristic tree ls into. It turns out that the class of racteristic trees generated by tree grammars is tained in the class of co-regular tree languages.

To show this fact, the following pages present construction method which takes any tree gramm produces a root-linear tree grammar C_G which g the set of characteristic trees CT_G.

Like an LR(0) parser, the method used to C_G is to create a new set of nonterminals usin "production slices" where a production slice i generalization of the concept of a marked prod

Given a tree grammar $G=(\Phi,\Sigma,P,S)$, a <u>produ</u> <u>slice</u> is any pair $(F(\vec{x})\text{->}t,U) \in P \times 2^{N^*}$ such tha following four conditions hold:

i) $F(\vec{x})\text{->}t \in P$

ii) $U \subseteq dom(t) \cup \{u0 \mid u \in const(t)\}$

iii) for all $u \in U$, there does not exist a v that v is a proper prefix of u

iv) for all $u \in (var(t) \cup \{u0 \mid u \in const(t)\})$ exists a $v \in U$ such that v is a prefix

Furthermore, let ps(P) be the set of all produ slices defined on the set of productions P. I words,

$ps(P) = \{(p,U) \mid (p,U)$ is a production slic

le 6.2.1: Let G=(Φ,Σ,P,S) be defined as in exampl
. Then,

```
(P) = {(S->F,{10}), (S->F,{1}), (S->F,{ε}),
           |               |               |
           a               a               a

       (F->F,{11}), (F->F,{1}), (F->F,{ε}),
        |  |            |  |            |  |
        x  f            x  f            x  f
           |               |               |
           x               x               x

       (F-> g ,{1,2}), (F-> g ,{ε})}.
        |  / \          |  / \
        x x   x         x x   x
```

The above production slices can be graphically
ted as follows (where the dots represent marked
ions):

```
-> F      S -> F      S -> .
   |           |           F
   a           .           |
   .           a           a

-> F      F -> F     F -> .
   |      |    |     |    F
   f      x    .     x    |
   |           f          f
   .           |          |
   x           x          x

->    g          F ->    .
   /     \       |       g
  .       .      x    /     \
  x       x           x      x
```

In using production slices as nonterminals, several patterns of reference reoccur in the fol pages. To simplify the burden of having to expr these patterns at each time, the following three definitions are presented:

<u>**Definition 6.2.1:**</u> For any tree grammar $G=(\bar{\Phi},\bar{\Sigma},P,$ the function $init_G : T_{\bar{\Sigma}\vee\bar{\Phi}}(\mathbf{X}_A) \to 2^{\mathbf{N}^*}$ be defined that for any tree $t \in T_{\bar{\Sigma}\vee\bar{\Phi}}(\mathbf{X}_A)$,
$init_G(t) = var(t) \vee \{u0 \mid u \in const(t)\}$.

<u>**Example 6.2.2:**</u> Let G be defined as in example 6. Then,

```
init (F)  = {10},  init (F)  = {11}, and
    G |                G |
      a                  f
                         |
                         x

init (g)  = {1,2}.
    G/ \
    x   x
```

<u>**Definition 6.2.2:**</u> Let $vn : 2^{\mathbf{N}^*} \times \mathbf{N}^* \to \mathbf{N}$ be a fu called the <u>variable</u> <u>name</u> <u>selector</u> and defined su for any $U \in 2^{\mathbf{N}^*}$, $u \in \mathbf{N}^*$, $vn(U,u)=|V|$ where
$V = \{v \in U \mid v \leq u, \exists w \in \mathbf{N}^* \text{ s.t. } w0=v\}$ and $\leq$ is the p lexicographical ordering for tree addresses.

The variable name selector takes a set of "do rom a production slice and a particular "dot" u i d returns the variable name that u represents. nstance, if vn(U,u)=i, then the "dot" u represent riable x_i. Note that a "dot" u does not corresp any variable if the "dot" u occurs below a leaf beled by a constant.

<u>**xample**</u> <u>**6.2.3:**</u> Let U={1·1·0, 1·2,1·3, 1·4·0, 2·1·0 ·2, 2·3, 3·0}. Then, by the above definition, n(U,1·1·0)=0, vn(U,1·2)=1, vn(U,1·3)=2, vn(U,1·4· n(U,2·1·0)=2, vn(U,2·2)=3, vn(U,2·3)=4, vn(U,3·0) nd vn(U,2)=2.

<u>**efinition**</u> <u>**6.2.3:**</u> Let vs : $2^{\mathbf{N}^*}$ -> **N** be a function alled the <u>variable</u> <u>size</u> <u>index</u> and defined such th or any U∈$2^{\mathbf{N}^*}$, vs(U) = {u∈U | ∃w∈$\mathbf{N}^*$ s.t. w0=u}.

Like the variable name selector, the variable ndex takes a set of "dots" U from a production sl he value returned is the number of variables the dots" of U represent. Since it is not necessaril ase that all "dots" represent variables (i.e. so

the dots may occur below leaves labeled
typically the variable size index of a s
same as the cardinality of the set.

Example 6.2.4: Let U be defined as in ex,
Then vs(U)=4.

Having provided the above helping f
root-linear tree grammar C^, is defined a

Definition 6.2*4: Given any tree grammar
G-CJJ.I^PJ.SJ) let $C_G=(1_2,I_2,P_2,S_2)$ be
characteristic grammar of G where

ijL = S_2Vps(P) where r(S_2)=0 and for e
production slice (p,U), r((p,U))ssv

1_2 - liVlj; and

P« is constructed as follows:

i) for every production of the for
S_2->N€P_2 where N-(S_1«->t, init$_G$(t

ii) for every nonterminal N€(jj • of t
(F(x)->t,{§}), N(x)->x€P_2

for every nonterminal $N_1 \in \bar{\Phi}_2$ of the form $(F(\vec{x}) \to t, U \vee \{u0\})$ such that $r(t(u))=0$, $vs(U \vee \{u0\})=i+n$, and $vn(U \vee \{u0\},u)=i$,
$N_1(x_1,\ldots,x_{i+n}) \to N_2(x_1,\ldots,x_i,t(u),x_{i+1},\ldots,x_{i+n}) \in P_2$ where $N_2=(F(\vec{x}) \to t, U \vee \{u\})$

for every nonterminal $N_1 \in \bar{\Phi}_2$ of the form $(F(\vec{x}) \to t, U \vee \{u1,\ldots,um\})$ such that $r(t(u))=m>0$, $vs(U)=i+n$ and $vn(U \vee \{u1,\ldots,um\},u1)=i+1$,
$N_1(x_1,\ldots,x_{i+n+m}) \to N_2(x_1,\ldots,x_i, t(u)(x_{i+1},\ldots,x_{i+n}), x_{i+n+1},\ldots,x_{i+n+m}) \in P_2$ where $N_2=(F(\vec{x}) \to t, U \vee \{u\})$

for every nonterminal $N_1 \in \bar{\Phi}_2$ of the form $(F(\vec{x}) \to t, U \vee \{u0\})$ such that $u \in dom(t)$, $t(u)=G \in \bar{\Phi}_1$, $r(G)=0$, $G \to s \in P_1$ and $vs(U)=k$,
$N_1(x_1,\ldots,x_k) \to N_2 \in P_2$ where $N_2=(G \to s, init_G(s))$

for every nonterminal $N_1 \in \bar{\Phi}_2$ of the form $(F(\vec{x}) \to t, U \vee \{u1,\ldots,um\})$ such that $u \in dom(t)$, $t(u)=G \in \bar{\Phi}_1$, $r(G)=m$, $vn(U \vee \{u1,\ldots,um\},u1)=i+1$, and $vs(U \vee \{u1,\ldots,um\})=k$, for each $G(\vec{x}) \to s \in P_1$,
$N_1(x_1,\ldots,x_k) \to N_2(x'_1,\ldots,x'_q) \in P_2$ where $N_2=(G(\vec{x}) \to s, V)$, $V=init_G(s)$, $vs(V)=q$, and for all $v \in V$ such that $v \in var(s)$, if $s(v)=x_j$ for

some j, $1 \leq j \leq m$, and $vn(V,v)=p$, the

vii) nothing else.

Note that condition (ii) states that
in a production slice have been moved to t
production slice (i.e. the root), then th
tree is a characteristic tree. Condition
the dot below the leaf u (in the productio
over the constant labeling the leaf. Conc
moves m dots immediately below the node u
production slice) up over the internal noc
t(u). Condition (v) simulates all possibl
derivation steps on a nonterminal labeling
condition (vi) simulates all possible OI d
steps on a nonterminal labeling an interna

Example 6.2.5: Let G be defined as in exam
Then $C_G=(\bar{\Phi}',\bar{\Sigma}',P',S')$ such that

$\bar{\Phi}' = \{S'\} \vee ps(P)$ where ps(P) is the s
shown in example 6.2.1;

$\bar{\Sigma}' = \{S,F,a,b,c\}$; and

P' is defined by the following producti

```
S′ -> (S->F)
          |
          a
          .

(S->.)(x) -> x
    F
    |
    x

(F->.)(x) -> x
 |  F
 x  |
    f
    |
    x

(F->.)(x) -> x
 |  g
 x / \
   x x

(S->F)(x) -> (S->F)(a)
    |            |
    a            .
    .            a

(S->F)(x) -> (S->.)(F)
    |            F  |
    .            |  x
    a            a

(F->F)(x) -> (F->F)(f)
 |  |         |  |  |
 x  f         x  .  x
    |            f
    .            |
    x            x

(F->F)(x) -> (F->.)(F)
 |  |         |  F  |
 x  .         x  |  x
    f            f
    |            |
    x            x
```

```
(F-> g )(x,y) -> (F-> . )(   g   )
 |  / \            |   g     / \
 x .   .           x  / \   x   y
   x   x              x x

(S->F)(x) -> (F->F)(x)
    |          |  |
    .          x  f
    x             |
                  .
                  x

(S->F)(x) -> (F-> g )(x,x)
    |          |  / \
    .          x .   .
    a            x   x

(F->F)(x) -> (F->F)(x)
 |  |          |  |
 x  .          x  f
    f             |
    |             .
    x             x

(F->F)(x) -> (F-> g )(x,x)}.
 |  |          |  / \
 x  .          x  . .
    f             x x
    |
    x
```

By the definition of G,

```
S ==> F ==> F ==> F ==>     g
  OI  |  OI |  OI |  OI   /   \
      a     f     f     f     f
            |     |     |     |
            a     f     f     f
                  |     |     |
                  a     a     a
```

Hence, g is a characteristic tree.

```
      g
    /   \
   f     f
   |     |
   f     f
```

corresponding derivation in C_G that generates the
ve characteristic tree is as follows:

```
S' =OI=> (S->F) =OI=> (S->F)(a) =OI=> (F->F)(a)
            |              |           |  |
            a              .           x  f
            .              a              |
                                          .
                                          x

=OI=> (F->F)(f) =OI=> (F->F)(f) =OI=> (F->F)(f)
       |  |  |         |  |  |         |  |  |
       x  .  a         x  f  a         x  .  f
          f               |               f  |
          |               .               |  a
          x               x               x

=OI=> (F->  g  )(f,f) =OI=> (F->   .   )(  g  )
       |   / \   | |          |    g      / \
       x  .   .  f f          x   / \    f   f
          x   x  | |             x   x   |   |
                 a a                     f   f
                                         |   |
                                         a   a

=OI=>  g
      / \
     f   f
     |   |
     f   f
     |   |
     a   a
```

Having introduced some notation, the following
ma shows that

```
(F-> f  )(a,a) =*OI=>  (F-> . )(  f  )
 |  / \                 |   f    / \
 x .   F                x  / \  a   F
   x   |                  x   F     |
       f                      |     f
       |                      f     |
                              |     a
```

In other words, one can move the "dots" up on a production slice, and the corresponding nodes th dots move over become terminal symbols in the de tree.

Lemma 6.2.1: Given any tree grammar $G=(\Phi_1,\Sigma_1,P_1,$ its corresponding characteristic grammar $C_G=(\Phi_2,\Sigma_2,P_2,S_2)$; any $n\geq 0$; any two terminals N such that $F(\vec{x})\text{->}t\in P_1$, $U\in 2^{\mathbf{N}^*}$, $u\in dom(t)$, $N_1=(F(\vec{x})\text{->}t,U\cup\{u\})$, and $N_2=(F(\vec{x})\text{->}t,U\cup V\}$ where $V \subseteq \{uw \mid w\in\mathbf{N}^*\}$ and $n=\max\{length(w) \mid uw\in V\}$; th $t'=\{(w,f) \mid (uw,f)\in t;\ \exists v\in V$ such that uw is a p prefix of $v\} \cup \{(w,x_i) \mid uw\in dom(t), uw\in V,$ and $vn(V,w)=i\}$; and any sequence of trees $t_1,\ldots,t_m$ such that $m=vs(U\cup V)$, then $N_2(t_1,\ldots,t_m) \overset{*}{\underset{OI}{\Longrightarrow}}{}_{C_G}$ $N_1(t_1,\ldots,t_i,t'(t_{i+1},\ldots,t_{i+p}),t_{i+p+1},\ldots,t_{i+p+q}$ $i+p+q=m$, $vs(V)=p$, and $vn(U\cup\{u\},u)=i+1$.

Proof: By induction on n.

base case: $n=0$ - Trivial.

inductive step: $N_1=(F(\vec{x})\text{->}t,U\cup\{u\})$ and $N_2=(F(\vec{x})\text{->}t,U\cup V)$ where $V \subseteq \{uw \mid w\in\mathbf{N}^*\}$ and $n+1=\max\{length(w) \mid uw\in V\}$. Depending on the ari t(u), there are two cases:

se 1: $V=\{u0\}$ where $t(u)=a\in\Sigma_1\vee\Phi_1$ and $r(t(u))=0$.
e definition of C_G, $N_2(x_1,\ldots,x_{i+q})$ ->
$(x_1,\ldots,x_i,a,x_{i+1},\ldots,x_{i+q})\in P_2$ where $vs(U\vee\{u0\})=$
d $vn(U\vee\{u\},u)=i+1$. Hence $N_2(t_1,\ldots,t_{i+q}) \xRightarrow[\overline{OI}]{}$
$(t_1,\ldots,t_i,a,t_{i+1},\ldots,t_{i+q})$.

se 2: $r(t(u))=p>0$. By the definition of C_G,
$(x_1,\ldots,x_{i+p+q})$ ->
$(x_1,\ldots,x_i,t(u)(x_{i+1},\ldots,x_{i+p}),x_{i+p+1},\ldots,x_{i+p+q})$
ere $vs(U)=i+q$, $vn(U\vee\{u1,\ldots,up\},u1)=i+1$, and
$=(F(\vec{x})\text{->}t,U\vee\{u1,\ldots,up\})$. Hence
$(t_1,\ldots,t_i,t'_1,\ldots,t'_p,t_{i+p+1},\ldots,t_{i+p+q}) \xRightarrow[\overline{OI}]{}$
$(t_1,\ldots,t_i,t'(t_{i+1},\ldots,t_{i+p}),t_{i+p+1},\ldots,t_{i+p+q})$ w
r all j, $1\leq j\leq p$, $t'_j=t'(t_{i+1},\ldots,t_{i+p})/j$. By
duction, for all j, $1\leq j<p$,
$(\vec{x})\text{->}t,U\vee V_j\vee W_j)(t_1,\ldots,t_i,\ t^j_1,\ldots,t^j_{p_j},$
$_{+p+1},\ldots,t_{i+p+q}) \xRightarrow[\overline{OI}]{*} (F(\vec{x})\text{->}t,U\vee V_{j+1}\vee W_{j+1})$
$_1,\ldots,t_i,\ t^{j+1}_1,\ldots,t^{j+1}_{p_{j+1}},\ t_{i+p+1},\ldots,t_{i+p+q})$ wher
$=\{v\in V \mid uk$ is a proper prefix of $v,\ j\leq k\leq p\}$,
$=\{uk \mid 1\leq k<j\}$, $p_j=vs(W_j\vee V_j)$, and for all k, $1\leq k\leq$

$$= \begin{cases} t'(t_{i+1},\ldots,t_{i+p})/k & \text{if } k<j \\ t_{i+k} & \text{otherwise.} \end{cases}$$

Before continuing, let me introduce the fo

definition:

Definition 6.2,5: Given any tree grammar

G=(li,Ai,Pi,Si) and its characteristic grammar

$C_G=(\Phi_2,\Sigma_2,P_2,S_2)$, any nonterminal N€f? of the f

(F($\vec{x}$)->t,U),* the production slice supertree, de

pss(N), is the tree defined by the set of pairs

{(w,f) | (w,f)€t; 3v€U s.t. w is proper pref

V {(w,x.) | w€U, 3fv€N* s.t. vO=w, and vn(U,w]

Example 6.2.6: Let G and C_G be defined as in ea

6.2.5. Then

```
pss(S->F) = F      pss(S->F) » F    pss(S->.) = u
          II                 II                 F
          a    a             .   x              |
          •                  a                  a

pss(F->F) = F      pss(F->F) = F    pss(F->.) = x
      i   1    1         1  1    1      1  F
      x   f    f         x  •    x      x  1
          1    1            f              f
          •    x            1              I
          x                 a              a

pss(F->  g  ) =  g     pss(F->  .  ) =  x
     I   / \    / \         I    g
     x   •   •  x  y        x   / \
         xx                     xx
```

Note; The following fact is important and is us

succeeding proofs. For any nonterminal N€(jj? of

form (F("$\vec{x}$)->t>init$_o$(t)), for any sequence of ti

$s_1,\ldots,s_{pi}, t_1,\ldots,t_q$ €Txr2^2 where r(F)=p and

$init_G(t))=q$, and for all $v \in var(t)$, if
$init_G(t),v)=i$, $t(v)=x_j$, and $t_i=s_j$, then $t(s_1,...$
$ss(N)(t_1,...,t_q)$.

The next lemma presents a slight extension of
nma 6.1.1 and states that given any production sl
the form $N=(F(\vec{x})\text{->}t,init_G(t))$, $N(\vec{x}) \overset{*}{\underset{OI}{\Longrightarrow}} pss(N)(\vec{x}$

<u>**nma 6.2.2:**</u> Given any tree grammar $G=(\bar{\Phi}_1,\bar{\Sigma}_1,P_1,S_1)$
s characteristic grammar $C_G=(\bar{\Phi}_2,\bar{\Sigma}_2,P_2,S_2)$; any
nterminal $N \in \bar{\Phi}_2$ of the form $(F(\vec{x})\text{->}t,U)$ where
$init_G(t)$; and any sequence of trees $t_1,...,t_m \in T_{\bar{\Sigma}}$
ere $vs(U)=m$, $N(t_1,...,t_m) \overset{*}{\underset{OI}{\Longrightarrow}}_{C_G} pss(N)(t_1,...,t_m)$

<u>**oof:**</u> By lemma 6.1.1, $N(t_1,...,t_m) \overset{*}{\underset{OI}{\Longrightarrow}}$
$(\vec{x})\text{->}t,\{\varepsilon\})(t'(t_1,...,t_m))$ where $t'=\{(w,f) \mid (w,f$
$v \in U$ s.t. w is a proper prefix of $v\} \vee$
$w,x_i) \mid w \in dom(t), w \in U, vn(U,u)=i\}$. By inspection
early $t'=pss(N)$. Hence $(F(\vec{x})\text{->}t,\{\varepsilon\})(t'(t_1,...,t$
$(F(\vec{x})\text{->}t,\{\varepsilon\})(pss(N)(t_1,...,t_m))$. By definition
, $(F(\vec{x})\text{->}t,\{e\})(x)\text{->}x \in P_2$. Hence
$(\vec{x})\text{->}t,\{\varepsilon\})(pss(t_1,...,t_m)) \underset{OI}{\Longrightarrow} pss(N)(t_1,...,t_m)$

Having presented the above two lemmas,
shows that for any tree $t \in CT_G$, $t \in L(C_G)$.

Lemma 6.2.3: Given a tree grammar $G=(\Phi_1,\Sigma_1,$
its characteristic grammar $C_G=(\Phi_2,\Sigma_2,P_2,S_2)$
$s[u\text{<-}F(t_1,\ldots,t_m)] \overline{\overline{OI}}{>}_G\ s[u\text{<-}t(t_1,\ldots,t_m)]$
$r(F)=m$, then

i) $S_2 \overset{*}{\overline{\overline{OI}}{>}}_{C_G} t(t_1,\ldots,t_m)$

ii) $S_2 \overset{*}{\overline{\overline{OI}}{>}}_{C_G} (F(\vec{x})\text{->}t,V)(t'_1,\ldots,t'_q)$ wh
$V=init_G(t)$, $q=vs(V)$, and for all v
there does not exist a $w \in \mathbf{N}^*$ where
$vn(V,v)=i$, $t(t_1,\ldots,t_m)/v=t'_i$.

Proof: By induction on n.

base case: $S_1 \overline{\overline{OI}}{>}_G\ t$. By definition of C_G
$S_2\text{->}(S_1\text{->}t,init_G(t)) \in P_2$. Hence, using lemm
$\overline{\overline{OI}}{>}_{C_G} (S_1\text{->}t,init_G(t)) \overset{*}{\overline{\overline{OI}}{>}}_{C_G} pss(t) = t$.

inductive step: $S_1 \overset{n+1}{\overline{\overline{OI}}{>}}_G s[u\text{<-}F(t_1,\ldots,t_m$
$s[u\text{<-}t(t_1,\ldots,t_m)]$. By the definition of $\overline{\overline{O}}$
must exist a derivation such that $S_1 \overset{n}{\overline{\overline{OI}}{>}}_G$
$s'[v\text{<-}G(s'_1,\ldots,s'_{q'})] \overline{\overline{OI}}{>}_G\ s'[v\text{<-}t'(s'_1,\ldots,s$
$s[u\text{<-}F(t_1,\ldots,t_m)]$ where $q'=r(G)$, there exi
such that $vw=u$ and $t'(s'_1,\ldots,s'_{q'})/w=F(t_1,\ldots$
for all proper prefixes y of u, $t(y) \in \Sigma_1$. B

$\overset{*}{\underset{\overline{OI}}{\Longrightarrow}}_{C_G}$ $(G(\vec{x})\text{->}t',V)(s_1,\ldots,s_q)$ where $V=init_G(t')$,
(V)=q, and for all $z\in V$ such that there does not e
$y\in\mathbf{N}^*$ where $y0=z$, and if $vn(V,z)=i$, then
$=t'(s_1',\ldots,s_{q'}')/z$. By lemma 6.1.1,
$(\vec{x})\text{->}t',V)(s_1,\ldots,s_q) \overset{*}{\underset{\overline{OI}}{\Longrightarrow}}_{C_G} (G(\vec{x})\text{->}t',W\vee\{w1,\ldots,$
$_1,\ldots,s_i,\ t_1,\ldots,t_m,\ s_{i+p+1},\ldots,s_{i+p+k})$ where
$\{v\in V \mid w$ is not a prefix of $v\}$, $vs(W)=p$, $i+p+k=q$,
$(W\vee\{w1,\ldots,wm\},w1)=i+1$. By the definition of C_G
$(\vec{x})\text{->}t,W\vee\{w1,\ldots,wm\})(x_1,\ldots,x_{i+k+m})$ ->
$(\vec{x})\text{->}t,Y)(x_1',\ldots,x_d')$ where $Y=init_G(t)$, $d=vs(Y)$, f
l $b\in Y$ where there does not exist a $c\in\mathbf{N}$ such that
$=b$, if $t(b)=x_j$ and $vn(Y,b)=p$, then $x_p'=x_{i+j}$. Henc
$(\vec{x})\text{->}t',W\vee\{w1,\ldots,wm\})$ $(s_1,\ldots,s_i,\ t_1,\ldots,t_m,$
$_{+p+1},\ldots,s_{i+p+k}) \underset{\overline{OI}}{\Longrightarrow}_{C_G} (F(\vec{x})\text{->}t,Y)(t_1',\ldots,t_d')$ whe
r all k, $1\le k\le d$, if $x_k'=x_{i+j}$, then $t_k'=t_j$. By lemma
2.2, $(F(\vec{x})\text{->}t,Y)(t_1',\ldots,t_d') \overset{*}{\underset{\overline{OI}}{\Longrightarrow}}_{C_G}$
$s(F(\vec{x})\text{->}t,Y)(t_1',\ldots,t_d')$. By inspection of the
finition a production slice supertree, clearly
$s(F(\vec{x})\text{->}t,Y)(t_1',\ldots,t_d') = t(t_1,\ldots,t_m)$.

To show inclusion in the reverse direction (i.
iat for all $t\in L(C_G)$, $t\in CT_G$ is not as easy. The m
oblem is that for any derivation in C_G, one must
with the derivation in G where it is not the ca
iat every derivation step in C_G corresponds to a

sequence of derivation steps in G. To aide in controlling this problem, one would like to ha method of picking out which derivation steps i correspond to derivation steps in G. The foll definitions provide this assistance by stating productions in C_G will correspond to derivatio in G.

Definition 6.2.6: Given any tree grammar $G=(\bar{\Phi}_1,\bar{\Sigma}_1,P_1,S_1)$ and its characteristic grammar $C_G=(\bar{\Phi}_2,\bar{\Sigma}_2,P_2,S_2)$, a <u>rewrite production</u> p is an production $p \in P_2$ such that p is of the form $N_1(x_1,\ldots,x_k) \rightarrow N_2(x'_1,\ldots,x'_q)$ where either

i) N_1 is of the form $(F(\vec{x})\rightarrow t, U\vee\{u\})$ su $u \in dom(t)$, $t(u)=G\in\bar{\Phi}_1$, $r(G)=0$, and $vs(U\vee\{u0\})=k$; and N_2 is of the form $(G(\vec{x})\rightarrow s,V)$ such that $V=init_G(s)$, and

ii) N_1 is of the form $(F(\vec{x})\rightarrow t, U\vee\{u1,\ldots$ such that $u \in dom(t)$, $t(u)=G\in\bar{\Phi}_1$, $r(G)=m$ $G(\vec{x})\rightarrow s \in P_1$, $vn(U\vee\{u1,\ldots,um\},u1)=i+1$ is of the form $(G(\vec{x})\rightarrow s,V)$ where $V=in$ $vs(V)=q$, and for all $v\in V$ such that $v\in$ if $s(v)=x_j$ for some j, $1\leq j\leq m$, and vn(then $x'_p=x_{i+j}$

Furthermore, let rewrite(P) denote the set of all productions p∈P such that p is a rewrite producti

Example 6.2.7: Let G and C_G be defined as in exam 6.2.5. Then rewrite(P_2) contains the following f productions:

```
(S->F)(x) -> (F->F)(x)
     |         |  |
     .         x  f
     a            |
                  .
                  x

(S->F)(x) -> (F->  g  )(x,x)
     |         |  / \
     .         x .   .
     a           x   x

(F->F)(x) -> (F->F)(x)
 |   |         |  |
 x   .         x  f
     f            |
     |            .
     x            x

(F->F)(x) -> (F->  g  )(x,x)
 |   |         |  / \
 x   .         x .   .
     f           x   x
     |
     x
```

Having defined which productions meet the de conditions, one can explicitly state which deriva steps use the rewrite productions, and this is presented by the following definition:

Definition 6.2.7: Given any tree grammar $G=(\Phi_1,\Sigma_1,P_1,S_1)$ and its characteristic grammar $C_G=(\Phi_2,\Sigma_2,P_2,S_2)$, a characteristic derivation s denoted $\xRightarrow{C}$, is an IO derivation step such that $t_1 \xRightarrow{C} t_2$ if and only if

i) $t_1 \underset{OI}{\Rightarrow}_{C_G} t_2$

ii) t_1 and t_2 are of the form
$t_1 = s[u\leftarrow F(s_1,\ldots,s_m)]$ and
$t_2 = s[u\leftarrow t(s_1,\ldots,s_m)]$ where
$F(\vec{x})\rightarrow t\in rewrite(P_2)$.

Similarly, a noncharacteristic derivation step, $\xRightarrow{NC}$, is an OI derivation step such that $t_1 \xRightarrow{NC}$ and only if $t_1 \underset{OI}{\Rightarrow}_{C_G} t_2$ and $t_1 \not\xRightarrow{C} t_2$.

Lemma 6.2.4 (below) shows that by only per noncharacteristic derivations, the derivation m performed on production slices defined on a sin production, and that each noncharacteristic der step moves "dots" up in the production slice.

Lemma 6.2.4: Given any tree grammar $G=(\Phi_1,\Sigma_1,P_1$ its characteristic grammar $C_G=(\Phi_2,\Sigma_2,P_2,S_2)$; a any two nonterminals $N_1,N_2\in\Phi_2$ of the form

=(F($\vec{x}$)->t,U) and N_2=(G($\vec{x}$)->s,V) where vs(U)=p and
(V)=q; any sequence of trees
,...,s_q,t_1,...,$t_p \in T_{\overline{\Sigma}_2}$; if $N_1(t_1,\ldots,t_p) \xrightarrow{NC}{}^n$
(s_1,...,s_q) then

i) F($\vec{x}$)->t = G($\vec{x}$)->s

ii) for all v∈V, there exists a u∈U such that
a prefix of u

iii) for all u∈U∧V such that there does not exi
a v∈$\mathbf{N}^*$ where v0=u, t_i=s_j where vn(U,u)=i a
vn(V,u)=j

iv) for all v∈V-(U∧V), s_i=t′(t_{j+1},...,t_{j+k}) wh
vn(V,v)=i, Y={w | vw∈U},
t′ = {(w,f) | (vw,f)∈t, ∃u∈U s.t. vw is
prefix of u} ∨ {(w,x_h) | vw∈dom(t), vw∈U,
vn(Y,w)=h}, vn(U,v)=j and vs(Y)=k

<u>oof:</u> By induction on n.

<u>se</u> <u>case:</u> n = 0, trivial.

<u>ductive</u> <u>step:</u> $N_1(t'_1,\ldots,t'_m) \xrightarrow{NC}{}^n N_2(t_1,\ldots,t_p)$ $\underline{N}$
(s_1,...,s_q). By induction,

i) N_j - (F(*)->t,U) and N_2 = (F($\vec{x}$)->t,V)
F(*)->t6P_1

ii) for all v€V, there exists a u€U such
a prefix of u

iii) for all u€UAV such that there does n
a v€N where vO=u, t'_i-t_j where vn(U,i
vn(V,u)=j

iv) for all v€V-(UAV), $t_\pm$ - t'(t' x_{j+}, . . .
where vn(V,v)=i, Y={w | vw€U}, t'={(*
(vw,f)€t; 3u€U s.t. vw is a proper
u} V {(w,x_h) ! vw€dom(t), vw€Y, vn(Y
vn(UV{v})=j, and vs(Y)=k.

By inspection of the definition of C , there a
two applicable forms of productions which will
noncharacteristic derivation:

<u>case 1;</u> N_2 = (F($\vec{x}$)->t,WV{uO}) and
N_3 = (F(ie)->t,WV{u}) where V=WV{uO}, r(t(u))=
vs(WV{uO})=i+j, vn(WV{u},u)=i+l, and N ^ x ^ . ,
-> N_3(x_1, . • . ,x_{jL}, t(u), x_{i+1}, . . . ,x_{i+j})• Hence
N_2(t_1,, . . ,t_p) ——> N_3(t_L, . . . ,t_i,t(u),$t_{\pm+1}$, . . . ,t
Clearly, condition (i) of the lemma is met. I
V=WV{uO} and u is a prefix of uO, clearly for
v€WV{u}, there exists a u€U such that v is a

Hence condition (ii) of the lemma is met. By
spection of the definition of set intersection,
$(W \vee \{u\})=U \wedge V$, and hence by the induction applied
rlier, condition (iii) of the lemma is met. Since
r all $v \in W-\{U \wedge W)$ and $s_i=\{(\varepsilon,t(u))\}=t'$, clearly
ndition (iv) is met.

se 2: $N_2 = (F(\vec{x})\text{->}t, W \vee \{u1,\ldots,um\})$ and
$= (F(\vec{x})\text{->}t, W \vee \{u\})$ where $V=W \vee \{u1,\ldots,um\}$,
$t(u))=m>0$, $vs(W)=i+j$, $vn(W \vee \{u1,\ldots,um\},u1)=i+1$, a
$(x_1,\ldots,x_{i+m+j}) \text{->} N_3(x_1,\ldots,x_i,\ t(u)(x_{i+1},\ldots,x_{i+}$
$_{m+1},\ldots,x_{i+m+j}) \in P_2$. Hence $N_2(t_1,\ldots,t_{i+m+j}) \xrightarrow{NC}$
$(t_1,\ldots,t_i,\ t(u)(t_{i+1},\ldots,t_{i+m}),\ t_{i+m+1},\ldots,t_{i+m+}$
early condition (i) of the lemma is met. Since
$W \vee \{u1,\ldots,um\}$ and u is a prefix of u1, clearly fo
l $v \in W \vee \{u1,\ldots,um\}$ there exists a $u \in U$ such that v
prefix of u. Hence condition (ii) of the lemma i
t. Also, $U \wedge (W \vee \{u\})=U \wedge V$, and hence by the
duction applied earlier, condition (iii) of the l
met. Therefore, the only condition left to show
at $s_{i+1}=t'(t'_{j+1},\ldots,t'_{j+m})$ where
$_{+1}=t(u)(t_{j+1},\ldots,t_{j+m})$, and $vn(W \vee \{u\},u)=i+1$ is
ready known. Let $Y=\{w \mid uw \in U\}$. CLearly, from th
ductive step applied earlier, for all h, $1 \leq h \leq m$, t
$t'(t'_{j+1},\ldots,t'_{j+k})/h$ where $vn(W \vee \{u\},u)=j$, $t'=\{(w,f$
$w,f) \in t$, $\exists v \in U$ s.t. uw is a prefix of v$\}$ $\vee$ $\{(w,x_c$

$uw\in dom(t)$, $uw\in U$, $vn(Y,w)=c\}$, $vn(u,v)=j$, and vs(

But then $s_{i+1} = t(u)(t'(t'_{j+1},\ldots,t'_{j+k}))/1, \ldots$

$t'(t'_{j+1},\ldots,t'_{j+k})/m) = t'(t'_{j+1},\ldots,t'_{j+k})$. Henc

condition (iv) of the lemma is met.

By separating derivations into characteris

noncharacteristic derivations, the following le

shows that for any tree $t\in L(C_G)$, $t\in CT_G$.

<u>**Lemma**</u> <u>6.2.5:</u> Given any tree grammar $G=(\bar{\Phi}_1,\bar{\Sigma}_2,P_1$

its characteristic grammar $C_G=(\bar{\Phi}_2,\bar{\Sigma}_2,P_2,S_2)$; a

any nonterminal $N_0=(S_1\text{->}t_0,init_G(t_0))\in\bar{\Phi}_2$; any

of nonterminals $N_1,\ldots N_n\in\bar{\Phi}_2$ such that for all i

$N_i=(F_i(\vec{x})\text{->}t_i,U_i)$ for some production $F_i(\vec{x})\text{->}t_i$

$S_2 \overset{}{\underset{\overline{OI}}{\Longrightarrow}} N_0 \overset{NC}{\Longrightarrow}{}^{p_0} N'_0(t'_{0,1},\ldots,t'_{0,m'_0}) \overset{C}{\Longrightarrow}$

$N_1(t_{1,1},\ldots,t_{1,m_1}) \overset{NC}{\Longrightarrow}{}^{p_1} N'_1(t'_{1,1},\ldots,t'_{1,m'_1}) \overset{C}{\Longrightarrow}$

$N_2(t_{2,1},\ldots,t_{2,m_2}) \overset{NC}{\Longrightarrow}{}^{p_2} \ldots \overset{C}{\Longrightarrow} N_n(t_{n,1},\ldots,$

$\overset{NC}{\Longrightarrow}{}^{p_n} N'_n(t'_{n,1},\ldots,t'_{n,m'_n}) \underset{\overline{OI}}{\Longrightarrow}{}_{C_G} t'$ where

i) $t'\in T_{\bar{\Sigma}_2}$

ii) for all i, $0\leq i\leq n$, $N'_i\in\bar{\Phi}_2$

iii) for all , $0 \leq i \leq n$, $p_i \geq 0$

iv) for all i, $0 \leq i \leq n$, $m_i = r(N_i)$ and $m'_i = r(N'_i)$

v) for all i, $0 \leq i \leq n$, all j, $1 \leq j \leq m_i$, all k, $1 \leq k \leq m'_i$, $t_{i,j}, t'_{i,k} \in T_{\Sigma_2}$

vi) there are exactly n characteristic deriva steps

hen there exists a derivation in G such that $S_1 \underset{\overline{OI}}{\Longrightarrow}$ $[u \text{<-} F_n(s_1,\ldots,s_p)] \underset{\overline{OI}}{\Longrightarrow}_G s[u \text{<-} t_n(s_1,\ldots,s_n)]$ where ss$(N_n)(t_{n,1},\ldots,t_{n,m_n}) = t_n(s_1,\ldots,s_p) = t'$.

roof: By induction on n.

ase case: n=0. Hence $S_2 \Longrightarrow N_0 \xRightarrow{NC} p_0$ $_0(t'_{0,1},\ldots,t'_{0,m'_0}) \underset{\overline{OI}}{\Longrightarrow} t'$. By inspection of the efinition of C_G, the last derivation step must be he form $N'_0(x) \text{->} x$ where N'_0 is of the form $F(\vec{x}) \text{->} t, \{\varepsilon\})$. By lemma 6.2.4, $N'_0 = (S_1 \text{->} t_0, \{\varepsilon\})$ ar $\xRightarrow{C} p_0$ $N'_0(t_0) \underset{\overline{OI}}{\Longrightarrow} t_0 = t'$. Clearly $S_1 \text{->} t_0 \in P_1$ and h $_1 \underset{\overline{OI}}{\Longrightarrow}_G t_0$.

nductive step: For any $n \geq 1$, $S_2 \underset{\overline{OI}}{\Longrightarrow} N_0 \xRightarrow{NC} p_0$ $'_0(t'_{0,1},\ldots,t'_{0,m'_0}) \xRightarrow{C} N_1(t_{1,1},\ldots,t_{1,m_1}) \xRightarrow{NC} p_1$ .. $\xRightarrow{C} N_n(t_{n,1},\ldots,t_{n,m_n}) \xRightarrow{NC} p_n$ $N'_n(t'_{n,1},\ldots,t'_{n,m'_n}) \underset{\overline{OI}}{\Longrightarrow}$ y inspection of the definition of C_G, the last erivation step must have been of the form $N'_n(x) \text{->}$

where N'_n is of the form $(F(\vec{x})\text{->}t,\{\varepsilon\})$. By the definition of $\xrightarrow{C}$, $N_n=(F_n(\vec{x})\text{->}t_n,U_n)$ where $U_n=init_G(t_n)$. By lemma 6.2.4, $N_n(t_{n,1},\ldots,t_{n,m_n})$ $N'_n(pss(N_n)(t_{n,1},\ldots,t_{n,m_n})$ $\overline{\overline{OI}}>$ $pss(N_n)(t_{n,1},\ldots,t_{n,m_n})$ = t' where $N'_n=(F_n(\vec{x})\text{->}t_n,$ Similarly, $N_{n-1}=(F_{n-1}(\vec{x})\text{->}t_{n-1},U_{n-1})$ where $U_{n-1}=init_G(t_{n-1})$ and $N_{n-1}(t_{n-1,1},\ldots,t_{n-1,m_{n-1}})$ $\underline{N}$ $N'_{n-1}(t'_{n-1},\ldots,t'_{n,m'_m})$ $\xrightarrow{NC}{}^{*}$ $N''_{n-1}(pss(N_{n-1})\ (t_{n-1,1},\ldots,t_{n-1,m_{n-1}}))$ $\overline{\overline{OI}}>$ $pss(N_{n-1})(t_{n-1,1},\ldots,t_{n-1,m_{n-1}})$ where N'_{n-1} = $(F_{n-1}(\vec{x})\text{->}t_{n-1},U'_{n-1})$ and $N''_{n-1}=(F_{n-1}(\vec{x})\text{->}t_{n-1},\{\varepsilon\})$ induction, there exists a derivation such that S_1 $s[v\text{<-}F_{n-1}(s_1,\ldots,s_p)]$ $\overline{\overline{OI}}>_G$ $s[v\text{<-}t_{n-1}(s_1,\ldots,s_p)]$ $p=r(F_{n-1})$ and $pss(N_{n-1})(t_{n-1,1},\ldots,t_{n-1,m_{n-1}})$ = $t_{n-1}(s_1,\ldots,s_p)$. From lemma 6.2.4, N'_{n-1} = $(F_{n-1}(\vec{x})\text{->}t_{n-1},U'_{n-1})$ where for all $w\in U'_{n-1}$, there a $y\in U_{n-1}$ such that w is a prefix of y. By the definition of $\xrightarrow{C}$ either

i) $U'_{n-1} = W \vee \{u1,\ldots,uq\}$

ii) $U'_{n-1} = W \vee \{u0\}$

and $t(u) = F_n$. Clearly, for all $w\in U_n$ such that $w\in var(t_n)$, if $t_n(w)=x_j$ and $vn(U,w)=p$, then $t_{n,p}$ = $t_{n-1}(s_1,\ldots,s_p)/uj$. Also, $s[v\text{<-}t_{n-1}(s_1,\ldots,s_p)]$ $s[v\text{<-}t_{n-1}(s_1,\ldots,s_p)\ [u\text{<-}F_n(t_{n-1}(s_1,\ldots,s_p)/u1,$.

$_{n-1}(s_1,\ldots,s_p)/uq)]] \overrightarrow{\overline{OI}}_G\ s[v\leftarrow t_{n-1}(s_1,\ldots,s_p)$ $u\leftarrow t_n(t_{n-1}(s_1,\ldots,s_p)/u1,\ \ldots,\ t_{n-1}(s_1,\ldots,s_p)/uq$ here $r(F_n)=q$. Hence $t_n(t_{n-1}(s_1,\ldots,s_p)/u1,\ \ldots,$ $_{n-1}(s_1,\ldots,s_p)/uq)) \in CT_G$. On the other hand, by nspection of the definition of a production slice upertree, $pss(N_n)(t_{n,1},\ldots,t_{n,m_n}) =$ $_n(t_{n-1}(s_1,\ldots,s_p)/u1,\ \ldots,\ t_{n-1}(s_1,\ldots,s_p)/uq))=t$ ence $t' \in CT_G$.

Having shown in lemma 6.2.3 that for all $t \in CT$ $\in L(C_G)$, and in lemma 6.2.5 that for all $t \in L(C_G)$, $\in CT_G$, the following two theorems states the desir esults, namely that $L(C_G)=CT_G$ and that the class haracteristic trees is contained in the class of oregular tree languages.

heorem 6.2.1: Given any tree grammar $G=(\Phi_1,\Sigma_1,P_1,$ nd its characteristic grammar $C_G=(\Phi_2,\Sigma_2,P_2,S_2)$, $(C_G) = CT_G$.

roof: By definition, $CT_G = \{t(t_1,\ldots,t_n)\ |\ S \overset{*}{\overrightarrow{\overline{OI}}}_G$ $[u\leftarrow F(t_1,\ldots,t_n)] \overrightarrow{\overline{OI}}_G\ s[u\leftarrow t(t_1,\ldots,t_n)]\}$ and $L($ $t \in T_{\Sigma_2}\ |\ S_2 \overset{*}{\overrightarrow{\overline{OI}}}_{C_G}\ t\}$. Let $t(t_1,\ldots,t_n) \in CT_G$ be any n the set CT_G. Hence, there exists a derivation he form $S_1 \overset{*}{\overrightarrow{\overline{OI}}}_G\ s[u\leftarrow F(t_1,\ldots,t_n)] \overrightarrow{\overline{OI}}_G$

$s[u \leftarrow t(t_1,\ldots,t_n)]$ for some production $F(\vec{x}) \rightarrow t$
lemma 6.2.3, $S_2 \overset{*}{\underset{OI}{\Longrightarrow}}_{C_G} t(t_1,\ldots,t_n)$. Hence
$t(t_1,\ldots,t_n) \in L(C_G)$ and $CT_G \subseteq L(C_G)$. On the ot
let $t' \in L(C_G)$ be any tree in the set $L(C_G)$. He
there must exist a derivation such that $S_2 \overset{*}{\underset{OI}{\Longrightarrow}}$
By the definition of C_G, and $\underset{OI}{\Longrightarrow}$, the derivati
be of the form $S_2 \underset{OI}{\Longrightarrow}_{C_G} s_1 \overset{NC}{\Longrightarrow}^* s_2 \overset{C}{\Longrightarrow} s_3 \overset{NC}{\Longrightarrow}^*$
$\ldots \overset{C}{\Longrightarrow} s_5 \overset{NC}{\Longrightarrow}^* s_6 \underset{OI}{\Longrightarrow}_{C_G} t'$. By lemma 6.2.5,
must exist a derivation of the form $S_1 \overset{*}{\underset{OI}{\Longrightarrow}}_G$
$s[u \leftarrow F(t_1,\ldots,t_n)] \underset{OI}{\Longrightarrow} s[u \leftarrow t(t_1,\ldots,t_n)]$ whe
$t(t_1,\ldots,t_n)=t'$. Hence $t' \in CT_G$ and $L(C_G) \subseteq CT_G$
Therefore $CT_G = L(C_G)$.

Theorem 6.2.2: The class of characteristic tre
contained in the class of coregular tree langu

Proof: Let $G=(\Phi_1,\Sigma_1,P_1,S_1)$ by any tree gramma
$C_G=(\Phi_2,\Sigma_2,P_2,S_2)$ be the characteristic gramma
By theorem 6.2.1, $L(C_G)=CT_G$. By inspection of
definition of C_G, clearly C_G is root-linear.
is contained in the class of coregular tree la

While the preceeding result has shown that the ss of characteristic trees is contained in the cl coregular tree languages, there was no descriptio the form of deterministic automata needed to ognize the class of characteristic trees. thermore, there is no known construction method ch guarantees to produce a deterministic automato recognize tree languages in the class of coregula e languages.

At this point, there appears to be two options tinuing to lift LR(0) parsing techniques. One ion is to invent a new construction method which l guarantee to produce a deterministic automaton set of characteristic trees. The other option i relax the constraints of only accepting racteristic trees such that the relaxation rantees that the construction method will produce erministic automaton. The former method was empted with little success. Therefore, the latte hod was chosen. In fact, the constraints were axed such that a bottom-up tree automaton could b lt, and then the result of Rabin and Scott[59] co used (this theorem states that every bottom-up tr omaton can be converted into a deterministic tom-up tree automaton).

One of the reasons for presenting the const method to build a characteristic grammar is that construction method provides insight as to why a bottom-up tree automaton can not be built to rec the set of characteristic trees. The definition production slice implicitly implies context betw subtrees (i.e. each "dot" in a production slice requires the corresponding context of the other the production slice). Hence, in designing the construction method which will build the charact automaton, the method will attempt to capture th "essence" of the production slices, used to defi characteristic grammar, without requiring the co used by production slices. In doing so, one sho note that the construction method will build an automaton which will recognize a superset of the language generated by the characteristic grammar also allow illegal stack configurations to be ac by the characteristic automaton).

The relaxation is to go back to using marke productions instead of production slices. Given grammar $G=(\Phi,\Sigma,P,S)$, a <u>marked</u> <u>production</u> is a pa $(F(\vec{x})\text{->}t,u) \in P \times \mathbf{N}^*$ where $F(\vec{x})\text{->}t \in P$ is a producti $u \in dom(t) \cup \{u0 \mid u \in const(t)\}$ is a marker denoting relative position of a read-head in recognizing ction. Furthermore, let mp(P) be the set of all ble marked productions defined by the set of ctions P.

le 6.2.8: Let G=(<I>,][,P , S) be the tree grammar ed in example 6.1.1. Then

```
(P) = {(S->F,10), (S->F,1), (S->F,S),
              1            1           1
              a            a           a

      (F->F 11), (F->F 1), (F->F,E),
        1  1'      1  1'     1  1
        x  f       x  f      x  f
           1          1         1
           x          x         x

      (F-> g ,1), (F-> g , 2), (F-> g ,6)}
        I  / \      I  / \       I  / \
        x  x  x     x  x  x      x  x  x
```

arked productions can be graphically depicted as ws:

```
-> F        S -> F        S -> .
   I             1             F
   a             .             i
   .             a             1
                               a

-> F        F -> F        F -0> .
   I        I    1        1     F
   f        X    .        x     1
   I             f              f
   x             1              1
                 x              x

->  g       F -> g        F ->  #
   / \      I   / \       I     g
  .   X     X  X   .      x    / \
  x                x          x   x
```

Using marked productions instead of produ
slices, the new construction method will creat
characteristic automaton in such a manner as t
the productions presented in definition 6.2.4
possible. The nondeterministic version of the
characteristic automaton is defined as follows

Definition 6.2.8: Given a tree grammar $G=(\Phi,\Sigma,$
the bottom-up tree automaton $NCG=(\Sigma\vee\Phi,C,\delta,S,\{$
the <u>nondeterministic version of the characteri</u>
<u>automaton</u> where

$C = mp(P) \vee \{S,F\}$ is the set of states, an

δ is defined as follows:

i) for all productions $F(\vec{x})\text{->}t\in P$, for all
$u\in const(t)$, $(F(\vec{x})\text{->}t,u0)\in\delta(S,\varepsilon)$

ii) for all productions $F(\vec{x})\text{->}t\in P$,
$F\in\delta((F(\vec{x})\text{->}t,\varepsilon),\varepsilon)$

iii) for all productions $F(\vec{x})\text{->}t\in P$, for all
$u\in const(t)$, $(F(\vec{x})\text{->}t,u)\in\delta((F(\vec{x})\text{->}t,u0),$

iv) for all productions $F(\vec{x})\text{->}t\in P$, for each
such that $r(t(u))=m>0$,
$(F(\vec{x})\text{->}t,u)\in\delta((k_1,\ldots,k_m),t(u))$ where f
$1\leq i\leq m$, $k_i=(F(\vec{x})\text{->}t,ui)$

v) for all productions $F(\vec{x})\to t \in P$, for each $ui \in dom(t)$ such that $t(u)=G \in \Phi$, $r(G)=m$, and $1\leq$ for each production of the form $G(\vec{x})\to s \in P$, f each $v \in var(s)$ such that $s(v)=x_i$, $(G(\vec{x})\to s,v) \in \delta((F(\vec{x})\to t,ui),\varepsilon)$

vi) nothing else

ample 6.2.9: Let G be the tree grammar defined in ample 6.1.1. Then, the nondeterministic version e characteristic automaton is the bottom-up tree tomaton $NCG=(\Sigma \cup \Phi, C, \delta, S, \{F\})$ where

$C = mp(P) \cup \{S,F\}$ such that

```
S -> F will be denoted as s1,
     |
     a
     .

S -> F will be denoted as s2,
     |
     .
     a

S -> . will be denoted as s3,
     F
     |
     a

F -> . will be denoted as s4,
|    F
x    |
     f
     |
```

```
F -> F
|    |
x    .
     f
     |
     x
```

will be denoted as s_5,

```
F -> F
|    |
x    f
     |
     .
     x
```

will be denoted as s_6,

```
F ->   .
|      g
x     / \
     x   x
```

will be denoted as s_7,

```
F ->   g
|     / \
x    .   x
     x
```

will be denoted as s_8, and

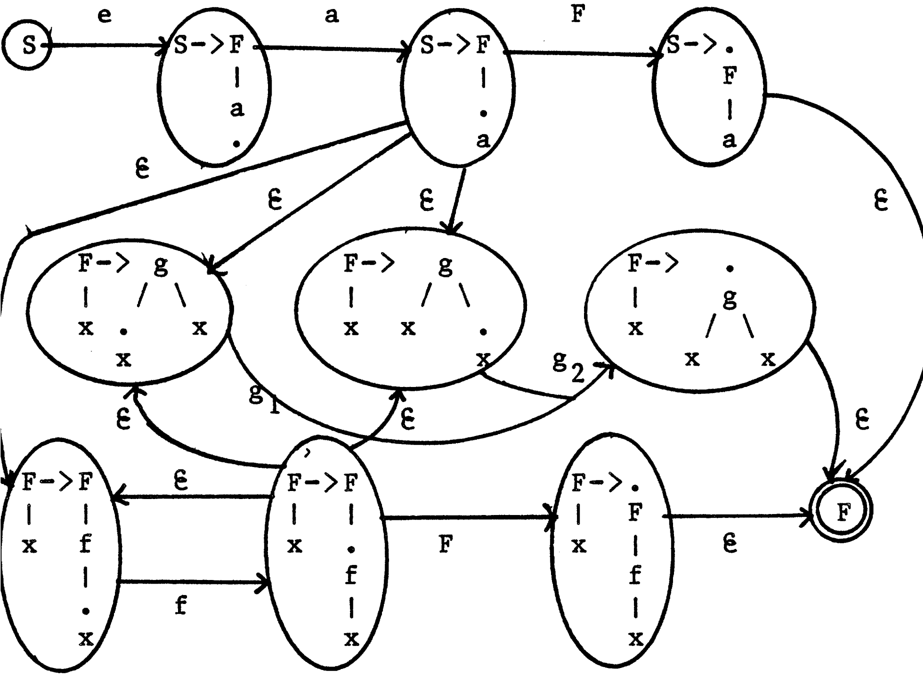
```
F ->   g
|     / \
x    x   .
         x
```

will be denoted as s_9; and

δ is defined by the following table.

	a	f	g	ε	F
S				$\{s_1\}$	
s_1	$\{s_2\}$				
s_2				$\{s_6, s_8, s_9\}$	
s_2)					$\{s_3\}$
s_3				$\{F\}$	
s_6)		$\{s_5\}$			
s_5				$\{s_6, s_8, s_9\}$	
s_5)					$\{s_4\}$
s_4				$\{F\}$	
s_7				$\{F\}$	
s_9)			$\{s_7\}$		

ote that δ can be graphically depicted as follows



As mentioned earlier, it has been shown by hatcher[73] that by using the construction of Rab nd Scott[59], every bottom-up tree automaton $=(\Sigma,C,\delta,q_0,F)$ can be converted to a deterministic ottom-up tree automaton $M'=(\Sigma,C'\delta',q_0',F')$ such th $(M)=N(M')$. Algorithm 6.2.1 (see below) presents onstruction method to build M′ and consists of th ain procedure "ITEMS", and two functions "closure GOTO". The basic idea used by the algorithm is t onstruct a bottom-up tree automaton M′ where M′

imultaneously follows every possible computation
y having each state $q' \in C'$ be a set of states in (
here q' is reachable in M', for an input tree t,
nd only if for all $q \in q'$, q is reachable in M usin
n terms of the algorithm, function "closure" perf
psilon-closure by taking a state $q' \in C'$ and return
he set of all states reachable from states in q'
ithout reading any more input. Procedure "ITEMS'
he main routine and starts by defining q_0' as the
psilon-closure of the start state q_0 in M. Then,
sing the function "GOTO", it takes each n-tuple
$q_1,\ldots,q_n) \in tuple_n(C')$ already built, and determi
he transitions as follows:

For each $f \in \Sigma$ where $r(f)=n$, if for all i, $1 \leq i \leq n$,
there exists a $q^i \in q_i$ such that $q \in \delta((q^1,\ldots,q^n)$,
then there is a unique transition in M' such th
$\delta'((q_1,\ldots,q_n),f)=q'$ where q' is the epsilon-cl
of the set $\{q \mid q \in \delta((q^1,\ldots,q^n),f),\ q^i \in q_i\}$.

nce the graph defining the transition map δ is bu
he set of final states F' is defined such that fo
very state $q' \in C'$, if there exists a state $q \in q'$ su
hat $q \in F$, then $q' \in F'$.

lgorithm 6.2.1: A method to construct a determini bottom-up tree automaton.

nput: a bottom-up tree automaton $M=(\Sigma,C,\delta,q_0,F)$

utput: a deterministic bottom-up tree automaton $M'=(\Sigma,C',\delta',q_0',F')$ where M' does not contain ar epsilon moves.

ethod: The three procedures below, initiated by calling ITEMS(M);

```
rocedure ITEMS(M);
  begin
    for all input pairs (a,b)∈tuples(C) x (Σ∪{ε
      initialize δ'(a,b) to ∅;
    q0' := closure({q0});
    C' := {q0'};
    repeat
      for each a∈Σ such that r(a)=0 do
        for each set q1∈C' do
          if q2=GOTO(q1,a) and q2≠∅
            then
              C' := C' ∪{q2};
              δ'(q1,a) := {q2};
            fi
        od
```

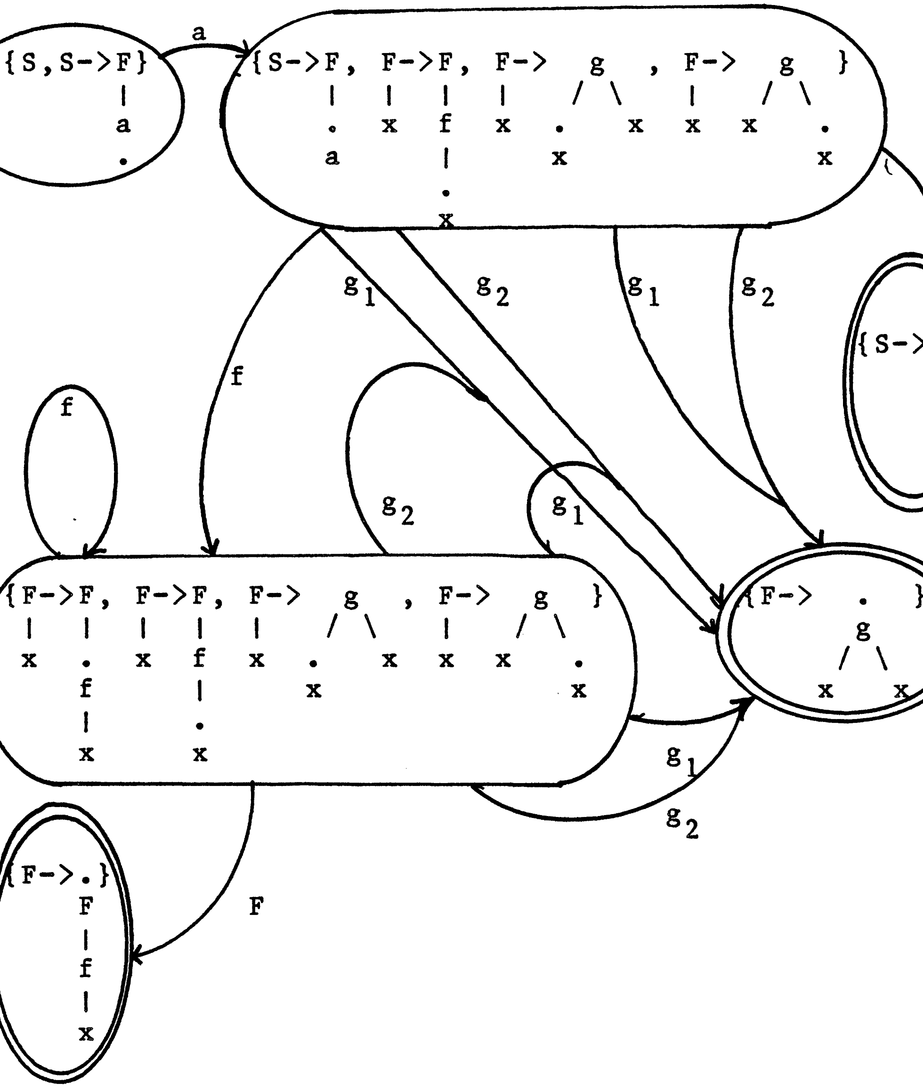
```
    for each f€i such that r(f)=»n>0 do
        for each n-tuple (q1,.*.,qn)€tuple n(C) dj
            if. GOTO((qlf ...fqn)ff)-q
                    where q^0
                then
                    C  := C V {q};
                    &'((qj,••.,qn),f) :=* {q};
                fi
            od
        od
until no more sets of states
        can be added to C';
F' := 0;
for each q'€C' dj3
    if there exists a q€q' such that q€F
        then F' :- F' Viq'}
    fi
od.
nd;
```

```
Function GOTO(z,f);
   begin
      if r(f)=0
         then
            q" := {q' | q'∈δ(q,f), q∈z}
         else
            q" := {q' | q'∈δ((q'_1,...,q'_n),f)
                        where r(f)=n, z is of the
                        (q_1,...,q_n), and for all
                        q'_i∈q_i}
         fi
      return closure(q")
   end
Function closure(k);
   begin
      s := k;
      while there exists a state q∈s such that
            q'∈δ(q,ε) and q'∉s
         do
            s := s ∨ {q'}
         od
      return s;
   end
```

<u>mple</u> <u>6.2.10:</u> Let NCG be the bottom-up tree automa
ined in example 6.2.9. Algorithm 6.2.1 will
struct the deterministic bottom-up tree automaton
$'=(\Sigma,C',\delta',q_0',F')$ where

```
C' = {{S, S->F}, {S->F, F->F, F->  g  , F->  g  }
             |       |  |  |  |   / \   |   / \
             a       .  x  f  x  .   x  x  x   .
             .       a     |     x             x
                           .
                           x

      {S->.}, {F->F, F->F, F->  g  , F->  g  },
          F    |  |  |  |  |   / \   |   / \
          |    x  .  x  f  x  .   x  x  x   .
          a       f     |     x             x
                  |     .
                  x     x

      {F->    .  }, {F->.}};
       |    g      |  F
       x   / \     x  |
          x   x       f
                      |
                      x

q0 = {S, S->F};
            |
            a
            .

F = {{S->.}, {F->   .  }, {F->.}}; and
         F    |    g      |  F
         |    x   / \     x  |
         a       x   x       f
                             |
                             x
```

δ is defined by the following graph:



Combining definition 6.2.8 and algorithm 6.2.1, BUTLR(0) characteristic automaton CG can be buil the tree grammar G. The previous method started structing the nondeterministic version of the racteristic automaton NCG using definition 6.2.8. n, NCG is made deterministic using algorithm 6.2. ducing the <u>BUTLR(0) characteristic automaton</u> CG.

ever, rather that going through the two step
cess, definition 6.2.8 and algorithm 6.2.1 can be
bined into a single algorithm as follows:

orithm 6.2.2: Method to construct a BUTLR(0)
characteristic automaton.

ut: a tree grammar $G=(\Phi,\Sigma,P,S)$

put: a deterministic bottom-up tree automaton
$CG=(\Sigma \vee \Phi, C, \delta, q_0, F)$ without epsilon-rules.

hod: The three procedures below, initiated
by calling ITEMS(G);

```
cedure ITEMS(G);
begin
   for all input pairs
        (a,b)∈tuples(C)) x (Σ∨{ε})
      initialize δ(a,b) := ∅;
   q0 := closure({(F(x⃗)->t,u0) |
                        F(x⃗)->t∈P, u∈const(t)};
   C := {q0};
```

```
repeat
   for each grammar symbol a∈Σ̄∨Φ̄
         such that r(a)=0 do
      for each set of marked productions I
         if J=GOTO(I,a) and J≠∅
            then
               C := C ∨ {J};
               δ(I,a) := {J};
            fi
         od
      od
   for each grammar symbol f∈Φ̄∨Σ̄
         such that r(f)=n>0 do
      for each n-tuple (I_1,...,I_n) in
            tuples(mp(P)) do
         if J=GOTO((I_1,...,I_m),f) and J≠∅
            then
               C := C ∨ {J};
               δ((I_1,...,I_n),f) := {J};
            fi
         od
      od
until no more sets of marked productions
      can be added to C;
```

```
    F := ∅;
    for each set of marked productions I∈C do
        if there exists a marked production of the
                form (F(x⃗)->t,ε)∈I
            then F := F ∨ {I}
        fi
    od
end

ıction GOTO(z,f);
begin
    if r(f)=0
        then
            J := {(F(x⃗)->t,u) |
                        (F(x⃗)->t,u0)∈z, t(u)=f};
        else
            J := {(F(x⃗)->t,u) | t(u)=f, r(t(u))=n,
                        z is of the form (I_1,...,I_n) an
                        for all i, 1≤i≤m, (F(x⃗)->t,ui)∈
        fi
    return closure(J)
end;
```

function closure(I);

    begin

        J := I;

        while there exists a marked production

                form $(F(\vec{x})\text{->}t,ui)\in J$ such that $i\in$

                $t(u)=G\in\Phi$ **do**

            for each $G(\vec{x})\text{->}s\in P$ **do**

                for each $v\in var(s)$ such that

                        $s(v)=x_i$, and $(G(\vec{x})\text{->}s,v)\notin J$

                do

                    $J := J \vee \{(G(\vec{x})\text{->}s,v)\}$;

                od;

            od

        od

        return J

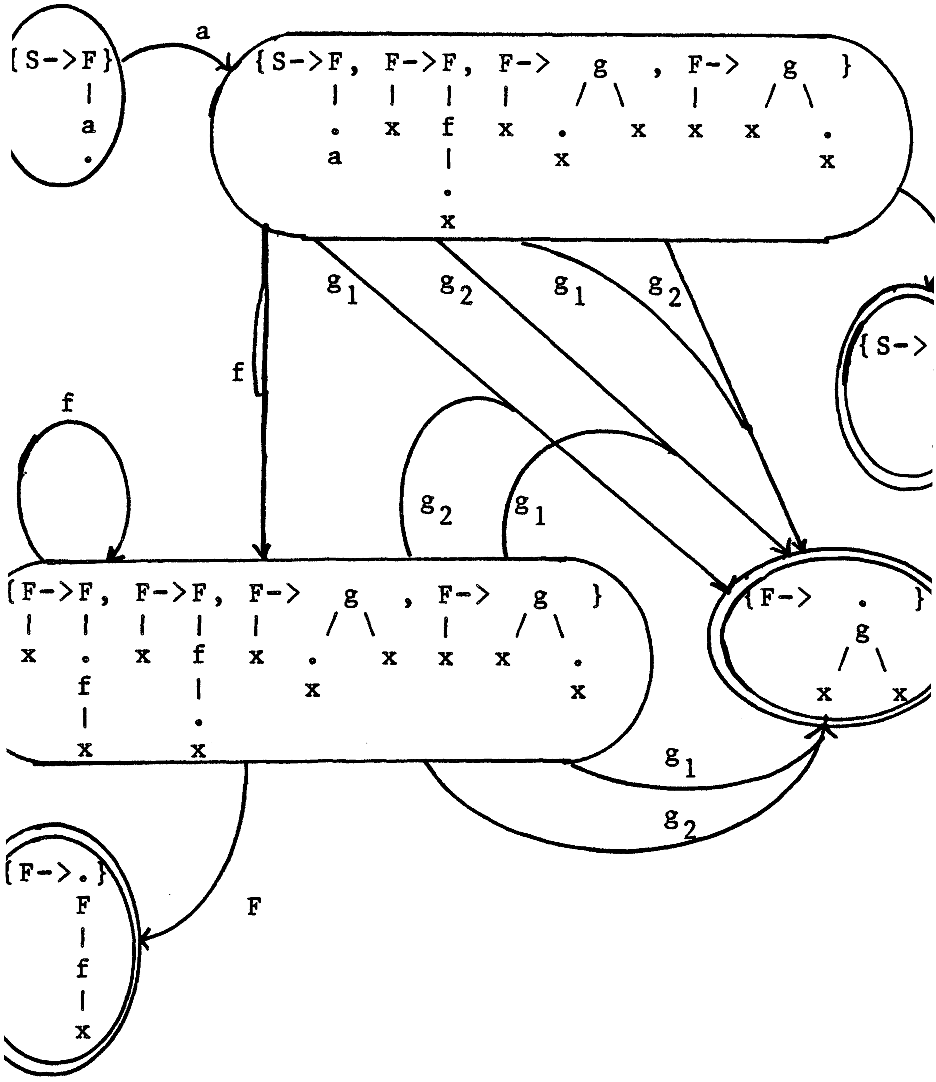
    end;

Example 6.2.11: Let $G=(\Phi,\Sigma,P,S)$ be the tree defined in example 6.2.1. Then, using algor the BUTLR(0) characteristic automaton is the tree automaton $CG=(\Sigma,C,\delta,q_0,F)$ where

```
C = {{S->F}, {S->F, F->F, F->   g  , F->
         |        |    |    |    / \     |    /
         a        .    x    f   x   .   x    x   x
         .        a         |       x
                            .
```

```
{S->.},  {F->F,  F->F,  F->   g   ,  F->   g   )
    F      I    I    I    I    I    / \    I    / \
    I          x  .  x  f  x  .        x  x  x    .
    a          £       I       x                  x
               |       .
               x       x

{F->   .   }, {F->.}};
 I     g           IF
 x    / \        x  |
     XX             f
                    |
                    x

qQ  -  {S->F};
           |
           a
           .

F = {{S->.},  {F->   .   },  {F->.}};  and
          F    i    g            I F
          I      x    / \          x  |
          a             x x         £
                                    |
                                    x
```

6 is defined by the following graph:



e that the only difference between CG and NCG′, sented in example 6.2.10, is that the start state tains a different set of states (i.e. the start te of NCG′ contains the state S which is not in C reason for this is that in creating NCG, a speci rt state S is added to give the nondeterministic omaton a single start state. However, when the

ph is made deterministic, the need for this state
oved and hence, algorithm 6.2.2 removes the state

This section concludes by showing that the abov
struction method is a strict relaxation on the
straint of only accepting the set of characterist
es. In other words, to show that for any
racteristic tree $t \in CT_G$, the characteristic automa
will accept t, which is shown by the following le
two theorems. One should also note that lemma
.6 shows the close correlation between the use of
ked productions and production slices.

ma 6.2.6: Given any tree grammar $G=(\Phi_1,\Sigma_1,P_1,S_1)$,
characteristic grammar $C_G=(\Phi_2,\Sigma_2,P_2,S_2)$, and its
racteristic automaton $CG=(\Sigma \vee \Phi, C, \delta, q_0, F)$; any $n \geq$
nonterminal $N \in \Phi_2$ where N is of the form $(F(\vec{x})\text{->}t$
$vs(U)=m$; any sequence of trees $t_1,\ldots,t_m \in T_{\bar{\Sigma}_2}$;
e $t' \in T_{\bar{\Sigma}_2}$ such that for any subset $Z \subseteq \{1,2,\ldots,m\}$
re $|Z|=q$, for all $i \in Z$ there exists a $w_i \in dom(t')$ s
t $t'=t'[w_i u_i \text{<-} t_i]$, $vn(U,u_i)=i$, and $u_i \in U$; any
$N^* \times C$; if $S_2 \Longrightarrow^n_{C_G} N(t_1,\ldots,t_m) \stackrel{*}{\Longrightarrow}_{C_G}$
$(N)(t_1,\ldots,t_m) = t'$, then there exist states
$\ldots,I_m \in C$ such that $(\{(w_i u_i v_i 0, q_0) \mid i \in Z,$
$\exists leaf(t_i)\} \vee b, t') \vdash^* (\{(w_i u_i, I_i) \mid i \in Z,$

$F(\vec{x})\text{->}t,u_i)\in I_i\}\vee b,t')$.

roof: By induction on n.

ase case: whenever n=1, clearly the above condit
re vacuously true.

nductive step: $S \Longrightarrow^n N_1(t'_1,\ldots,t'_{n_1}) \Longrightarrow$
$_2(t_1,\ldots,t_{n_2})$. Depending on the production used
he last derivation step, there are 4 cases:

ase 1 - condition (iii) of definition 6.2.4:
$F(\vec{x})\text{->}t,W\vee\{u0\})(t_1,\ldots,t_j, t_{j+2},\ldots,t_{n_2}) \Longrightarrow$
$F(\vec{x})\text{->}t,W\vee\{u\})(t_1,\ldots,t_{n_2})$ where $t_{j+1}=t/u$,
$(t(u))=0$, $U=W\vee\{u\}$, $vs(U)=n_2$, and $vn(U,u)=j+1$. By
efinition of the conditions of the lemma, there e
set $Z \subseteq \{1,2,\ldots,n_2\}$ such that $|Z|=q$ and for all
here exists a $w_i\in dom(t)$ such that $t'=t'[w_iu_i\text{<-}t_i]$
here $u_i\in U$ and $vn(U,u_i)=i$. If $j+1\in Z$, then by
nduction, there exists states $I_1,\ldots,I_{n_2}\in C$ such t
or all $i\in Z$, $(F(\vec{x})\text{->}t,u_i)\in I_i$ and $(\{(w_iu_iv_i0,q_0) \mid$
$_i\in leaf(t_i)\}\vee b,t') \vdash^* (\{(w_iu_i,I_i) \mid i\in Z, i\neq j+1,$
$F(\vec{x})\text{->}t,u_i)\in I_i\} \vee \{(w_{j+1}u_{j+1}0,q_0)\}\vee b,t')$. By th
onstruction of CG, clearly $(F(\vec{x})\text{->}t,u0)\in q_0$. By
nspection of the function GOTO, there exists an I
uch that $(F(\vec{x})\text{->}t,u)\in I_{j+1}$ and $I_{j+1}\in\delta(q_0,t(u))$. H
$\{(w_iu_i,I_i) \mid i\in Z, i\neq j+1, (F(\vec{x})\text{->}t,u_i)\in I_i\} \vee$
$(w_{j+1}u_{j+1}0,q_0)\}\vee b,t') \vdash (\{(w_iu_i,I_i) \mid i\in Z,$
$F(\vec{x})\text{->}t,u_i)\in I_i\}\vee b,t')$. On the other hand, if j+1

en by induction there exists states $I_1,\ldots,I_{n_2} \in C$
at for all $i \in Z$, $(F(\vec{x})\text{->}t,u_i) \in I_i$ and
$\{w_i u_i v_i 0, q_0) \mid i \in Z,\ v_i \in leaf(t_i)\} \vee b, t') \vdash^*$
$\{w_i u_i, I_i) \mid i \in Z,\ (F(\vec{x})\text{->}t,u_i) \in I_i\} \vee b, t')$.

se 2 - condition (iv) of definition 6.2.4:
$(t_1,\ldots,t_j,\ t_{j+1}/1,\ldots,t_{j+1}/p,\ t_{j+2},\ldots,t_{n_2}) \Longrightarrow$
$(t_1,\ldots,t_{n_2})$ where $N_1=(F(\vec{x})\text{->}t, W \vee \{u1,\ldots,up\})$,
$=(F(\vec{x})\text{->}t, W \vee \{u\})$, $vs(W \vee \{u\})=n_2$, $vn(W \vee \{u\},u)=j+1$,
l $r(t(u))=p$. By definition of the conditions of
mma, there exists a set $Z \subseteq \{1,2,\ldots,n_2\}$ such tha
$|=q$ and for all $i \in Z$, there exists a $w_i \in dom(t)$ suc
at $t'=t'[w_i u_i \text{<-} t_i]$ where $u_i \in W \vee \{u\}$ and
$(W \vee \{u\},u_i)=i$. If $j+1 \in Z$, then by induction there
ists states $I_1,\ldots,I_{n_2},I'_1,\ldots,I'_p \in C$ such that for
Z, $(F(\vec{x})\text{->}t,u_i) \in I_i$ and for all i, $1 \leq i \leq p$,
$(\vec{x})\text{->}t,ui) \in I'_i$, and $(\{(w_i u_i v_i 0, q_0) \mid i \in Z,$
$\in leaf(t_i)\} \vee b, t') \vdash^* (\{(w_i u_i, I_i) \mid i \in Z,\ i \neq j+1,$
$(\vec{x})\text{->}t,u_i) \in I_i\} \vee \{(w_{j+1} ui, I'_i) \mid 1 \leq i \leq p,$
$(\vec{x})\text{->}t,ui) \in I'_i\} \vee b, t')$. By inspection of the funct
TO in the construction of CG, there exists an I_{j+}
ch that $GOTO(I'_1,\ldots,I'_p),t(u))=I_{j+1}$ and
$(\vec{x})\text{->}t,u) \in I_{j+1}$. Hence, $I_{j+1} \in \delta((I'_1,\ldots,I'_p),t(u))$
$(w_i u_i, I_i) \mid i \in Z,\ i \neq j+1,\ (F(\vec{x})\text{->}t,u_i) \in I_i\} \vee$
$w_{j+1} u_{j+1} i, I'_i) \mid 1 \leq i \leq p,\ (F(\vec{x})\text{->}t,ui) \in I'_i\} \vee b, t') \vdash$
$(w_i u_i, I_i) \mid i \in Z,\ (F(\vec{x})\text{->}t,u_i) \in I_i\} \vee b, t')$. On the

other hand, if $j+1 \notin Z$, then by induction there ex
states $I_1,\ldots,I_{n_2} \in C$ such that for all $i \in Z$, $(F(\vec{x})\text{->}t,u_i) \in I_i$ and $(\{(w_iu_iv_i0,q_0) \mid i \in Z, v_i \in leaf(t_i)\} \vee b,t') \vdash^* (\{(,w_iu_i,I_i) \mid i \in Z, (F(\vec{x})\text{->}t,u_i) \in I_i\} \vee b,t')$.

<u>case</u> <u>3</u> - condition (v) of definition 6.2.4:
$N_1(t'_1,\ldots,t'_{n_1}) \Longrightarrow N_2$ where $r(N_2)=0$. Since ther
no parameters for N_2, the conditions of the lemm
vacuously true.

<u>case</u> <u>4</u> - condition (vi) of definition 6.2.4:
$N_1(t'_1,\ldots,t'_{n_1}) \Longrightarrow N_2(t_1,\ldots,t_{n_2})$ where $N_1=(F(\vec{x})\text{->}t,W\vee\{u1,\ldots,up\})$, $t(u)=G \in \Phi_1$ where $r(G$
$vn(W\vee\{u1,\ldots,up\},u1)=j+1$, $N_2=(G(\vec{x})\text{->}s,V)$ where
$V=init_G(s)$, $vs(V)=n_2$, and for all $v \in V$ such that
$v \in var(s)$, if $s(v)=x_k$ for some k, $1 \leq k \leq p$, and $vn(V$
then $t_q=t'_p$. By definition of the conditions of
lemma, for some subset $Z \subseteq \{1,2,\ldots,n_2\}$, for all
there exists a $w_i \in dom(t)$ such that $t'=t'[w_iu_i$<-t
where $u_i \in V$ and $vn(V,u_i)=i$. By induction (applie
many times as there are duplications of variable
occurring in the tree s) there exists states
$I_1,\ldots,I_p \in C$ such that for all $i \in Z$, $(F(\vec{x})\text{->}t,uk) \in$
where $s(u_i)=x_k$, and $(\{(w_iu_iv_i0,q_0) \mid i \in Z, v_i \in leaf(t_i)\} \vee b,t') \vdash^* (\{(w_iu_i,I_k) \mid i \in Z,$

$F(\vec{x})\text{->}t,uk)\in I_k$, $s(u_i)=x_k\}\vee b,t')$. By inspection o

he construction of CG, every state is closed usin

unction "closure". By inspection of the function

closure", clearly for all $i\in Z$, $(G(\vec{x})\text{->}t,u_i)\in I_k$ wh

$F(\vec{x})\text{->}uk)\in I_k$ and $s(u_i)=x_k$. For all $i\in Z$, let $I_i'=I$

here $s(u_i)=x_k$. But then $(\{(w_iu_i,I_k) \mid i\in Z,$

$F(\vec{x})\text{->}t,uk)\in I_k,\ s(u_i)=x_k\}\vee b,t') = (\{(w_iu_i,I_i') \mid i$

$G(\vec{x})\text{->}s,u_i)\in I_i'\}\vee b,t')$.

heorem 6.2.3: Given any tree grammar $G=(\Phi,\Sigma,P,S)$

ts characteristic automaton $CG=(\Sigma\vee\Phi,C,\delta,q_0,F)$, if

$\in CT_G$, then $t\in N(CG)$. That is, $CT_G \subseteq N(CG)$.

roof: By theorem 6.2.1, $L(C_G)=CT_G$ where

$_G=(\Phi_1,\Sigma_1,P_1,S_1)$ is the characteristic grammar of

ence, for any tree $t\in CT_G$, there exists a derivati

$\overset{*}{\Rightarrow}_{C_G}\ N(t) \Longrightarrow_{C_G}\ t$ where N is of the form

$F(\vec{x})\text{->}t,\{\varepsilon\})$. By lemma 6.2.6,

$\{(u0,q_0) \mid u\in leaf(t)\},t) \vdash^*$

$\{(\varepsilon,I) \mid (F(\vec{x})\text{->}t,\{\varepsilon\})\in I\},t)$. By the definition

G, $F=\{I\in C \mid (F(\vec{x})\text{->}t,\{\varepsilon\})\in I\}$. Therefore $t\in N(CG)$.

heorem 6.2.4: Given any tree grammar G, its set o

haracteristic trees CT_G, and its characteristic

utomaton CG, it is not necessarily the case that

$T_G = N(CG)$.

Proof: Assume that it is the case that for any t grammar G, CT_{Car} = N(CG). Consider the tree gramm shown in example 6•2•11, and its characteristic automaton CG. By theorem 6.2.3, $CT_{Car} \subseteq$ N(CG). B inspection of the created characteristic automat clearly g(f(f(a)),f(a))€N(CG). However, as show theorem 6.2.1, g(f(f(a)),f(a))£CT_{Car}• Hence CT_{KJ} C Therefore the set of characteristic trees CT_{CJ} is identical to the set of trees accepted by characteristic automaton CG, producing a contrad

Note; The result of theorem 6.2.4 is not surpris since not all coregular languages are accepted b bottom-up automata.

6.3 Constructing BUTLR(O) Parsing Tables

This section presents a method to construct BUTLR(O) parser from the BUTLR(O) characteristic automaton, and shows the correctness of the algo which creates the BUTLR(O) parser. The construe methods does not always construct a well defined BUTLR(O) parser. However, the produced BUTLR(O) always accepts the same tree language as the tre language generated by the given tree grammar, ev though the parser might be nondeterministic.

The method to convert the characteristic autom
to a BUTLR(0) parser is straight forward and is g
the following algorithm:

gorithm 6.3.1: Constructing a BUTLR(0) parser

put: a tree grammar $G=(\Phi,\Sigma,P,S)$ and its characteristic automaton $CG=(\Sigma\vee\Phi,C,\delta,q_0,F)$

tput: a BUTLR(0) parser $M=(G,K,\underline{shift},\underline{reduce},\underline{goto},\underline{start})$

thod: Let $C=\{I_1,I_2,\ldots,I_n\}$ be the set of sets of
rked productions from the characteristic automat
. Then, $K=\{1,2,\ldots,n\}$ where state i corresponds
e set of marked productions I_i, and $\underline{start}$=k where
$=q_0$ is the start state of the characteristic
tomaton CG. The rank of state i is determined by
ansition map δ of the characteristic automaton.
,

$$i) = \begin{cases} 0 \text{ if } i=\underline{start} \\ r(f) \text{ where } f\in\Sigma\vee\Phi \text{ and } I_i\in\delta(z,f) \\ \quad \text{for some } z\in\text{tuples}(C)\} \text{ otherwise.} \end{cases}$$

rthermore, the three parsing tables are construc
follows:

<u>shift</u> <u>table:</u>

i) for all states $k \in K$, all productions $F(\vec{x})$-
for all $u \in dom(t)$, if $t(u)=a \in \overline{\Sigma}$ where $r(a)$
$(F(\vec{x})\text{->}t,u) \in I_k$, and $I_j \in \delta(I_k,a)$, then
<u>shift</u>$(k,a)=j$

ii) for all n-tuples of states
$(k_1,\ldots,k_n) \in tuples(K)$, all productions
$F(\vec{x})\text{->}t \in P$, for all $u \in dom(t)$, if $t(u)=f \in \overline{\Sigma}$
$r(f)=n>0$, $(F(\vec{x})\text{->}t,ui) \in I_{k_i}$ for all i, $1 \leq$
and $I_j \in \delta((I_{k_1},\ldots,I_{k_n}),f)$, then
<u>shift</u>$((k_1,\ldots,k_n),a)=j$

iii) all other entries, not defined by (i) an
(ii), are defined as <u>error</u>

<u>reduce</u> <u>table:</u>

for all states $k \in K$,
<u>reduce</u>$(k)=\{(F(\vec{x})\text{->}t \mid (F(\vec{x})\text{->}t,\varepsilon) \in I_k\}$

<u>goto</u> <u>table:</u>

i) for all states $k \in K$, all productions $F(\vec{x})$-
for all $u \in dom(t)$, if $t(u)=A \in \overline{\Phi}$ where $r(A)$
$(F(\vec{x})\text{->}t,u) \in I_k$, and $I_j \in \delta(I_k,A)$, then
<u>goto</u>$(k,A)=j$

ii) for all n-tuples of states $(k_1,\ldots,k_n)\in$tuples(K), all productions $F(\vec{x})\text{->}t\in P$, for all $u\in dom(t)$, if $t(u)=F\in\Phi$ wh $r(F)=n>0$, $(F(\vec{x})\text{->}t,ui)\in I_{k_i}$ for all i, $1\leq i\leq n$ and $I_j\in\delta((I_{k_1},\ldots,I_{k_n}),F)$, then <u>goto</u>$((k_1,\ldots,k_n),F)=j$

iii) all other entries, not defined by (i) and (ii), are defined as <u>error</u>

<u>mple</u> <u>6.3.1:</u> Let $G=(\Phi,\Sigma,P,S)$ and its characteristi omaton $CG=(\Sigma,C,\delta,q_0,F)$ be defined as in example .11. Using algorithm 6.3.1, the constructed LR(0) parser is M=(G,K,<u>shift</u>,<u>reduce</u>,<u>goto</u>,<u>start</u>) re

K = {1,2,3,4,5,6} where r(1)=r(2)=0, r(3)=r(4)=r(6)=1, and r(5)=2; and

```
I1 = {S->F},
         |
         a
         .

I2 = {S->F, F->F, F->  g , F->  g },
         |     | |   / \    |  / \
         .     x f  .   x   x x   .
         a       | x                x
                 .
                 x
```

```
    I3 = {S->.},
             F
             |
             a

    I4 = {F->F, F->F, F->  g  , F->  g  },
          |  |  |  |  |   / \   |   / \
          x  .  x  f  x  .   x  x  x   .
             f     |     x             x
             |     .
             x     x

    I5 = {F->  .  }, and
          |    g
          x   / \
             x   x

    I6 = {F->.};
          |  F
          x  |
             f
             |
             x
```

and the *shift*, *reduce*, and *goto* functions are defined by the following tables:

```
           shift               reduce         goto

         a   f   g                              F
       +---+---+---+     +--------+       +---+
     1 | 2 |   |   |   3 |{ S->F }|   (2) | 3 |
       +---+---+---+     |     |  |       +---+
   (2) |   | 4 |   |     |     a  |   (4) | 6 |
       +---+---+---+     +--------+       +---+
 (2,2) |   |   | 5 |   5 |{F-> g }|
       +---+---+---+     | |  / \ |
 (2,4) |   |   | 5 |     | x x   x|
       +---+---+---+     +--------+
 (4,2) |   |   | 5 |   6 |{ F->F }|
       +---+---+---+     |  |  |  |
 (4,4) |   |   | 5 |     |  x  f  |
       +---+---+---+     |     |  |
   (4) |   | 4 |   |     |     x  |
       +---+---+---+     +--------+
```

te that the BUTLR(0) parser M corresponds to the

(0) parser presented in example 6.1.2.

The remainder of this section provides the
sary proofs in order to show that for any BUTLR(0
r M generated from a tree grammar G, the tree
age accepted by the BUTLR(0) parser M is identica
e tree language generated from the tree grammar G
result is shown in the following manner. First,
ne with TPDAs and STPDAs, lemmas 6.3.1 and 6.3.2
that every computation, using the decision
ion $\vdash_d$, can be converted to a computation under a
ix ordering. Then, lemma 6.3.3 shows the
lation between derivation steps using the tree
ar G, and the computation moves in the BUTLR(0)
r. Lemma 6.3.4 shows the relationship between
stack symbols and grammar symbols (known as the
ling" of the states). Using the definition of
ling", lemma 6.3.5 shows that the spelling of
tree stack will derive the corresponding portion
e input tree already scanned. Finally, Theorem
concludes this section by using the above result
ow the desired result that the tree language
ted by the BUTLR(0) M is identical to the tree
age generated by the tree grammar G.

In order to define a computation under a ordering, some terminology has to be introduce an *updated read-head of a computation* $id_1 \vdash_d$ i BUTLR(0) parser (denoted URH($id_1 \vdash_d id_2$)) be a the form $(u,p) \in \mathbb{N}^* \times T_K$ where id_1 and id_2 are instantaneous descriptions in SID, $id_1 \vdash_d id_2$, id_1, id_2 are in one of the following forms:

i) $id_1 = (\{(u0,\underline{start})\} \vee b, t)$ and

$id_2 = (\{(u,k)\} \vee b, t)$ where $r(k)=0$ and

ii) $id_1 = (\{(ui,p_i) \mid 1 \leq i \leq m\} \vee b, t)$ and

$id_2 = (\{(u,k(p_1,\ldots,p_m))\} \vee b, t)$ where

and $p=k(p_1,\ldots,p_m)$

iii) $id_1 = (\{(u,\beta(t_1,\ldots,t_m)\} \vee b, t)$ and

$id_1 = (\{(u,k(t_1,\ldots,t_m))\} \vee b, t)$ where

and $p=k(t_1,\ldots,t_m)$.

A *computation with postfix lower bound u* BUTLR(0) parser is the relation $\vdash_d^u \subseteq$ SID x SID such that for any two instantaneous descriptio and id_2, $id_1 \vdash_d^u id_2$ if and only if $id_1 \vdash_d id_2$ URH($id_1 \vdash_d id_2$)=(v,p) for some $(v,p) \in \mathbb{N}^* \times T_K$ w ("$\leq$" is the postfix lexicographical ordering f addresses).

Similarly, a computation under a postfix order r a BUTLR(0) parser is defined such that for any nputation $id_1 \vdash_d id_2 \vdash_d \ldots \vdash_d id_n$, if

i) for all i, $1 \leq i < n$, $URH(id_i \vdash_d id_{i+1}) = (u_i, p_i$
where $(u_i, p_i) \in N^* \times T_K$

ii) for all i, $1 \leq i \leq n$, for all j, $i \leq j \leq n$, $u_i \leq u_j$
where $\leq$ is the postfix lexicographical ordering for tree addresses

en, $id_1 \vdash_d id_2 \vdash_d \ldots \vdash_d id_n$ is a computation ur oostfix ordering. Also, whenever a computation $_1 \vdash_d id_2 \vdash_d \ldots \vdash_d id_n$ is a computation under a stfix ordering, it will be denoted as $id_1 \vdash_d^1 id_2$. $\vdash_d^1 id_n$, or simply $id_1 \vdash_d^{1\ n-1} id_n$.

Using these definitions, the following lemmas esented.

mma 6.3.1: Given a BUTLR(0) parser (G,K,shift,reduce,goto,start); any three stantaneous descriptions $id_1, id_2, id_3 \in SID$, for any N^*; any $n \geq 0$; if $id_1 \vdash_d^{u\ n} id_2 \vdash_d id_3$ where $H(id_2 \vdash_d id_3) = (v,p)$ for some $(v,p) \in N^* \times T_K$ and v< nder a postfix lexicographical ordering for tree dresses), then there exists an instantaneous scription id_2' such that $id_1 \vdash_d id_2' \vdash_d^{u\ n} id_3$ where

$URH(id_1 \vdash id_2') = (v,p)$.

Proof: Analogous to lemma 5.3.I.

Lemma 6.3.2; Given a BUTLR(O) parser M^s(G,K,shift,reduce,goto,start), any two insta descriptions $id_1, id_9 \in SID$, $id_1 \vdash_d^n id_2$ if and o $id_1 \vdash^n id_2$.

Proof; Analogous to lemma 5«3.2«

Since every computation can be converted computation under a postfix ordering, the rema this thesis will assume that all computations under a postfix ordering unless explicitly sta otherwise.

To show that the construction of the BUTL parser is correct, lemma 6«3#3 (below) starts showing that for any tree t in the tree langua generated by a tree grammar G, t is also accep its corresponding BUTLR(O) parser M. In other it shows the correlation between performing n derivation steps in G and the corresponding mo by the BUTLR(O) parser M.

ma 6.3.3: Given any tree grammar $G=(\Phi,\Sigma,P,S)$, its racteristic automaton $CG=(\Sigma \vee \Phi, C, \delta, q_0, F)$ where $I_1,\ldots,I_c\}$, and its BUTLR(0) parser $G,K,$shift,reduce,goto,start) where $K=\{1,\ldots,c\}$; $^* \times T_K$; any $t \in T_{\Sigma}$; any $n \geq 0$; any $F_1(\vec{x}) \rightarrow t_1 \in P$ wh $_1)=m$; if

i) $S \overset{*}{\underset{OI}{\Longrightarrow}}_G s[u \leftarrow F_1(s_1,\ldots,s_m)] \underset{OI}{\Longrightarrow}_G$
$s[u \leftarrow t_1(s_1,\ldots,s_m)] \overset{*}{\underset{OI}{\Longrightarrow}}_G$
$s[u \leftarrow t_2'(s_1,\ldots,s_m)[z \leftarrow t_1/v(s_1,\ldots,s_m)]] \overset{n}{\underset{OI}{\Longrightarrow}}$
$s[u \leftarrow t_2'(s_1,\ldots,s_m)[z \leftarrow t_2/z(s_1,\ldots,s_m)]] \overset{*}{\underset{OI}{\Longrightarrow}}$
$s[u \leftarrow t_2(s_1,\ldots,s_m)] \overset{*}{\underset{OI}{\Longrightarrow}}_G t$ where

a) $t_1/v \overset{n}{\underset{OI}{\Longrightarrow}}_G t_2/z$ (note that v is related z in the sense that once the supertree v is rewritten using an OI derivation t remove all nonterminals occurring as ancestors to the node v, the result of rewrites is that the tree t_1/v now occu at tree address z)

b) for all $w \in dom(t_2)-var(t_2)$, $uw \in dom(t)$ an $t_2(w)=t(uw)$,

c) for all $w \in dom(t_2')$ such that w is

of z, $w \in dom(t_2)$ and $t_2(w)=t_2'(w)$

ii) there exists trees $s_1',\ldots,s_m' \in T_K$ such

all i, $1 \leq i \leq m$, $s_i' \in \{\beta \mid \beta \in skeleton(s_i)$,

where for all $w \in var(t_1)$ such that $t_1($

$(F_1(\vec{x})->t_1,w) \in I_{k_i}$

then there exists a tree $s' \in skeleton(t_1/v)$ suc

for all $w \in dom(t_1/v)$, $(F_1(\vec{x})->t_1,vw) \in I_{s'(s_1',\ldots}$

and $(\{(uzw0,\underline{start}) \mid zw \in const(t_2)\} \vee$

$\{(uzw,s_i') \mid zw \in var(t_2),\ t_2(zw)=x_i\} \vee b,t) \vdash_d^*$

$(\{(uz,s'(s_1',\ldots,s_m'))\} \vee b,t)$.

Proof: By induction on the pair $(n,depth(t_1/v))$

the lexicographical ordering "$\leq$" where $(a,b) \leq ($

and only if either $a<c$, or $a=c$ and $b \leq d$.

base case: $n=0$ and $depth(t_1/v)=0$. Hence $r(t_1$

and $t_1(v)$ can not be a nonterminal. Depending

whether the one node tree is labeled by a vari

terminal constant, there are two cases:

case 1: $t_1(v)=x_i$ for some i, $1 \leq i \leq m$. By defin

$x_i \in skeleton(t_1/v)$. Hence $(\{(uz,s_i')\} \vee b,t) \vdash_d^*$

$(\{(uz,x_i(s_1',\ldots,s_m'))\} \vee b,t)$.

se 2: $t_1(v)=a \in \Sigma$ where $r(a)=0$. By inspection of nstruction of the characteristic automaton CG, $(\vec{x})\text{->}t,v0) \in I_{\underline{start}}=q_0$ and $GOTO(I_{\underline{start}},a)=I_j$ where d $(F(\vec{x})\text{->}t,v) \in I_j$. Hence $I_j \in \delta(I_{\underline{start}},a)$. By spection of the construction of the BUTLR(0) pars j=<u>shift</u>(<u>start</u>,a). Therefore $(\{(uz0,\underline{start})\}\vee,b,$ $(\{(uz,j)\}\vee b,t)$.

ductive step: $(0,0)<(n,depth(t_1/v)$. Depending e symbol labeling $t_1(v)$, there are three cases:

se 1: $t_1(v)=G \in \Phi$ where $r(G)=0$. By the definition OI derivation, it must be the case that the rivation is of the form $S \overset{*}{\underset{OI}{\Longrightarrow}}_G s[u\text{<-}F_1(s_1,\ldots,s_m)$ $\Rightarrow_G s[u\text{<-}t_1(s_1,\ldots,s_m)] \overset{*}{\underset{OI}{\Longrightarrow}}_G$ $u\text{<-}t_2'(s_1,\ldots,s_m)[z\text{<-}t_1/v(s_1,\ldots,s_m)]] =$ $u\text{<-}t_2'(s_1,\ldots,s_m)[z\text{<-}G]] \underset{OI}{\Longrightarrow}_G$ $u\text{<-}t_2'(s_1,\ldots,s_m)[z\text{<-}t']] \underset{OI}{\Longrightarrow}_G^{n-1}$ $u\text{<-}t_2'(s_1,\ldots,s_m)[z\text{<-}t_2/z]] \overset{*}{\underset{OI}{\Longrightarrow}}_G s[u\text{<-}t_2(s_1,\ldots,s$ $\Rightarrow_G$ t for some production $G\text{->}t' \in P$. By induction, $(uzw0,\underline{start}) \mid zw \in const(t_2)\}\vee b,t) \vdash_d^*$ $(uz,s')\}\vee b,t)$ where $s' \in skeleton(t')$ and for all $\in dom(t')$, $(G\text{->}t',w) \in I_{s'(w)}$. By the construction e BUTLR(0) parser M, $G\text{->}t' \in \underline{reduce}(I_{s'(\varepsilon)})$ since $\text{->}t',\varepsilon) \in I_{s'(\varepsilon)}$. Furthermore, by the construction , $(F(\vec{x})\text{->}t,v0) \in I_{\underline{start}}$ and $\delta(I_{\underline{start}},G)=I_k$ where

$F(\vec{x})\text{->}t,v)\in I_k$. By the construction of M, $=\underline{goto}(\underline{start},G)$. Therefore, by the definition of $\{(uz,s')\}\vee b,t) \vdash_d (\{(uz,k)\}\vee b,t)$.

ase 2: $t_1(v)=f\in\Sigma$ where $r(f)=p>0$. By induction, ll i, $1\leq i\leq p$, $(\{(uzjw0,\underline{start}) \mid i\leq j\leq p$, $zjw\in const(t$ $\{(uzjw,s'_d) \mid i\leq j\leq p$, $zjw\in var(t_2)$, $t_2(zjw)=x_d\} \vee$ $(uzj,s''_j(s'_1,\ldots,s'_m)) \mid 1\leq j<i\}\vee b,t) \vdash_d^*$ $\{(uzjw0,\underline{start}) \mid i<j\leq p$, $zjw\in const(t_2)\} \vee$ $(uzjw,s'_d) \mid i<j\leq p$, $zjw\in var(t_2)$, $t_2(zjw)=x_d\} \vee$ $(uzj,s''_j(s'_1,\ldots,s'_m)) \mid 1\leq j\leq i\}\vee b,t)$ where for all $\in dom(t_1/vi)$, $(F_1(\vec{x})\text{->}t_1,viw)\in I_{s''_i(s'_1,\ldots,s'_m)(w)}$. ll i, $1\leq i\leq p$, let $k_i=s''_i(s'_1,\ldots,s'_m)(\epsilon)$. By inspect f the construction of the characteristic automato $_k\in\delta((I_{k_1},\ldots,I_{k_p}),f)$ where $(F_1(\vec{x})\text{->}t_1,v)\in I_k$ and $_k=GOTO((I_{k_1},\ldots,I_{k_p}),f)$. By inspection of the onstruction of the BUTLR(0) parser M, $=\underline{shift}((k_1,\ldots,k_p),f)$. Therefore $\{(uzi,s''_i(s'_1,\ldots,s'_m)) \mid 1\leq i\leq p\}\vee b,t) \vdash_d$ $\{(uz,k(s''_1(s'_1,\ldots,s'_m),\ldots,s''_p(s'_1,\ldots,s'_m)))\}\vee b,t)$. urthermore, by inspection of the definition of keleton, clearly $k(s''_1,\ldots,s''_p)\in skeleton(t_1/v)$ and ll $w\in dom(t_1/v)$,

$F_1(\vec{x})\text{->}t_1,vw)\in I_{k(s''_1(s'_1,\ldots,s'_m),\ldots,s''_p(s'_1,\ldots,s'_m))}$

3: $t_1(v)=G\in\overline{\Phi}$ where $r(G)=p>0$. By the definition
an OI derivation, it must be the case that the
ivation is of the form $S \overset{*}{\underset{OI}{\Rightarrow}}_G s[u\leftarrow F_1(s_1,\ldots,s_m)]$
; $s[u\leftarrow t_1(s_1,\ldots,s_m)] \overset{*}{\underset{OI}{\Rightarrow}}_G$
$\leftarrow t_2'(s_1,\ldots,s_m)[z\leftarrow t_1/v(s_1,\ldots,s_m)]] =$
$\leftarrow t_2'(s_1,\ldots,s_m)[z\leftarrow G(t_1/v1(s_1,\ldots,s_m),$
$,t_1/vp(s_1,\ldots,s_m))]] \underset{OI}{\Rightarrow}_G$
$\leftarrow t_2'(s_1,\ldots,s_m)[z\leftarrow t'(t_1/v1(s_1,\ldots,s_m),$
$,t_1/vp(s_1,\ldots,s_m))]] \underset{OI}{\Rightarrow}_G^{n-1}$
$\leftarrow t_2'(s_1,\ldots,s_m)[z\leftarrow t_2/z(s_1,\ldots,s_m)]] \overset{*}{\underset{OI}{\Rightarrow}}_G$
$\leftarrow t_2(s_1,\ldots,s_m)] \overset{*}{\underset{OI}{\Rightarrow}}_G t$ for some production
$)\rightarrow t'\in P$. Let $var(t')=\{w_1,\ldots,w_q\}$ where $|var(t')|$
for all i, $1\leq i\leq q$, all j, $1\leq j\leq q$, $w_i\leq w_j$ where $\leq$ is
prefix lexicographical ordering of tree addresse
thermore, for all j, $1\leq j\leq q$, let $i_j=k$ where
$_j)=x_k$. Then, by the definition of an OI
ivation, it must be the case that the derivation
the form $S \overset{*}{\underset{OI}{\Rightarrow}}_G s[u\leftarrow F_1(s_1,\ldots,s_m)] \underset{OI}{\Rightarrow}_G$
$\leftarrow t_1(s_1,\ldots,s_m)] \overset{*}{\underset{OI}{\Rightarrow}}_G$
$\leftarrow t_2'(s_1,\ldots,s_m)[z\leftarrow t_1/v(s_1,\ldots,s_m)]] =$
$\leftarrow t_2'(s_1,\ldots,s_m)[z\leftarrow G(t_1/v1(s_1,\ldots,s_m),$
$,t_1/vp(s_1,\ldots,s_m))]] \underset{OI}{\Rightarrow}_G$
$\leftarrow t_2'(s_1,\ldots,s_m)[z\leftarrow t'(t_1/v1(s_1,\ldots,s_m),$
$,t_1/vp(s_1,\ldots,s_m))]] \underset{OI}{\Rightarrow}_G^{n_1}$
$\leftarrow t_2'(s_1,\ldots,s_m)[z\leftarrow t''_{2,1}(s_1,\ldots,s_m)$

$w_1 \texttt{<-} t_1/vi_1(s_1,\ldots,s_m)]\ [z_{w_2} \texttt{<-} t_1/vi_2(s_1,\ldots,s_m)]\ .$

$w_q \texttt{<-} t_1/vi_q(s_1,\ldots,s_m)]]]\ \overline{\overline{OI}}\!\!>_G{}^{n_2}$

$u \texttt{<-} t_2'(s_1,\ldots,s_m)[z \texttt{<-} t''_{2,2}(s_1,\ldots,s_m)$

$w_1 \texttt{<-} t_2/z_{w_1}(s_1,\ldots,s_m)]\ [z_{w_2} \texttt{<-} t_1/vi_2(s_1,\ldots,s_m)]$

$w_q \texttt{<-} t_1/vi_q(s_1,\ldots,s_m)]]]\ \overline{\overline{OI}}\!\!>_G{}^{n_3}\ \ldots\ \overline{\overline{OI}}\!\!>_G{}^{n_{q+1}}$

$u \texttt{<-} t_2'(s_1,\ldots,s_m)[z \texttt{<-} t''_{2,q}(s_1,\ldots,s_m)$

$w_1 \texttt{<-} t_2/z_{w_1}(s_1,\ldots,s_m)]\ \ldots$

$w_q \texttt{<-} t_2/z_{w_q}(s_1,\ldots,s_m)]]] =$

$u \texttt{<-} t_2'(s_1,\ldots,s_m)[z \texttt{<-} t_2/z(s_1,\ldots,s_m)]]\ \overset{*}{\overline{\overline{OI}}\!\!>}_G$

$u \texttt{<-} t_2(s_1,\ldots,s_m)]\ \overset{*}{\overline{\overline{OI}}\!\!>}_G\ t$ where $n_1+n_2+\ldots+n_{q+1}=n-1$

r all i, $1 \le i \le q$, for all $y \in dom(t''_{2,i})$ such that y i

efix of z_{w_i}, $y \in dom(t_2/z)$ and $t_2(zy)=t''_{2,i}(y)$. Her

induction, for all i, $1 \le i \le q$,

$(uzz_{w_j}w0,\underline{start})\ |\ i \le j \le q,\ zz_{w_j}w \in const(t_2)\} \vee$

$uzz_{w_j}w,s_d')\ |\ i \le j \le q,\ zz_{w_j}w \in var(t_2),\ t_2(zz_{w_j}w)=x_d\}$

$uzz_{w_j},s''_k(s_1',\ldots,s_m'))\ |\ 1 \le j < i,\ t'(w_j)=x_k\} \vee$

$uzw0,\underline{start})\ |\ 1 \le j \le q,\ zw \in const(t_2),$

a prefix y of w s.t. $y=z_{w_j}\} \vee b,t)\ \vdash_d^*$

$(uzz_{w_j}w0,\underline{start})\ |\ i < j \le q,\ zz_{w_j}w \in const(t_2)\} \vee$

$uzz_{w_j}w,s_d')\ |\ i < j \le q,\ zz_{w_j}w \in var(t_2),\ t_2(zz_{w_j}w)=x_d\}$

$uzz^{w_j},s''_k(s_1',\ldots,s_m'))\ |\ 1 \le j \le i,\ t'(w_j)=x_k\} \vee$

$uzw0,\underline{start})\ |\ 1 \le j \le q,\ zw \in const(t_2),$

a prefix y of w s.t. $y=z_{w_j}\} \vee b,t)$ where for all

$\in var(t')$ where $t'(w)=x_k$, $s''_k \in skeleton(t_1/vk)$ and fo

l $y \in dom(t_1/vk)$, $(F(\vec{x}) \texttt{->} t,vky) \in I_{s''_k(s_1',\ldots,s_m')}(y)$.

i, $1\leq i\leq p$, such that there does not exist a
ar(t') such that $t'(w)=x_i$, let $s''_i \in T_K$ be any tree
h that $s''_i \in$skeleton(t_1/vi) and for all $y\in$dom(t_1/v
$(\vec{x})\text{->}t_1, viy) \in I_{s''_i(s'_1,\ldots,s'_m)(y)}$. Furthermore, fo
i, $1\leq i\leq p$, let $k_i=s''_i(s'_1,\ldots,s'_m)(\varepsilon)$. By inspecti
the definition of the characteristic automaton C
ry state in C is closed using the function
osure". By inspection of the definition of the
ction "closure", for all i, $1\leq i\leq p$, for all $y\in$var
h that $t'(y)=x_i$, $(G(\vec{x})\text{->}t', y)\in I_{k_i}$. Hence, by
luction, ({(uzw0,<u>start</u>) | $1\leq i\leq q$, zw$\in$const(t_2), $\exists$
efix y of w such that $y=z_{w_i}$} $\vee$
$zz_{w_i}, s''_k(s'_1,\ldots,s'_m)$ | $1\leq i\leq q, t'(w_i)=x_k\} \vee b, t) \vdash^*_d$
$\{uz, s'(s''_1(s'_1,\ldots,s'_m),\ldots,s''_p(s'_1,\ldots,s'_m))\} \vee b, t)$ whe
skeleton(t') and for all $y\in$dom(t'),
$(\vec{x})\text{->}t', y)\in I_{s'(s''_1(s'_1,\ldots,s'_m),\ldots,s''_p(s'_1,\ldots,s'_m))(y}$
$k=s'(s''_1(s'_1,\ldots,s'_m),\ldots,s''_p(s'_1,\ldots,s'_m))(\varepsilon)$. By t
struction of the BUTLR(0) parser M,
$\vec{x})\text{->}t' \in$<u>reduce</u>(I_k). By the construction of the
racteristic automaton, $\delta((I_{k_1},\ldots,I_{k_p}),G)=I_{k'}$, wh
$(x)\text{->}t_1, v)\in I_{k'}$, and GOTO$((I_{k_1},\ldots,I_{k_p}),G)=I_{k'}$.
ce, by the construction of the BUTLR(0) parser M
<u>to</u>$((k_1,\ldots,k_p),G)=k'$. Therefore
$\{uz, s'(s''_1(s'_1,\ldots,s'_m),\ldots,s''_1(s'_1,\ldots,s'_m)))\} \vee b, t) \vdash$
$\{uz, k'(s''_1(s'_1,\ldots,s'_m),\ldots,s''_p(s'_1,\ldots,s'_m)))\} \vee b, t)$

meeting the conditions of the lemma.

In order to show that any tree accepted b
BUTLR(O) parser is generated by the correspond
grammar, the relationship between the states o
BUTLR(O) parser, and the grammar symbols of th
corresponding tree grammar, must first be esta
In other words, one must know what tree of gra
symbols a tree stack corresponds to© Like in
parsing, the link between grammar symbols and
done using the notion called the "spelling", a
defined by the following definition:

Definition 6.3.1; Given any tree grammar G*($,
its characteristic automaton CG=(]EV$, C , o, q_n , F
O{Ij, ..«,I $_c$}, and its BUTLR(O) parser
M=(G,K,shift,reduce,goto,start) where K-{1,..«
the relation spelling £ K x (2V3!) be defined \
for any k€K, any f€īVi, k spelling f if and 01
either

i) k = start and f=S where S is the start
the tree grammar G

$r(f)=0$ and $I_k \in \delta(q_0, f)$

$r(f)=m>0$ and there exists states $k_1, \ldots, k_m \in K$ such that $I_k \in \delta((I_{k_1}, \ldots, I_{k_m}), f)$

e next lemma present a crucial result by showing e relation spelling is a total function.

.3.4: Given any tree grammar $G=(\Phi, \Sigma, P, S)$, its eristic automaton $CG=(\Sigma \vee \Phi, C, \delta, q_0, F)$, and its) parser M=(G,K,shift,reduce,goto,start), the n spelling is a total function.

Assume that spelling is not a function. Then,

there exists a $k \in K$ such that there does not exist an $f \in \Sigma \vee \Phi$ where k spelling f.

there exists a $k \in K$ such that for some two symbols $f,g \in \Sigma \vee \Phi$ where $f \neq g$, k spelling f, and k spelling g.

the first case has to be false since by the ction of the set of states C, $q_0 \in C$, and $I_k \in C$ there is a transition, defined on some grammar to the state I_k. Hence, in order for the

relation spelling not to be a function, it mus
case that there exists a state $k \in K$ such that k
f and k spelling g where $f \neq g$. By inspection o
definition of spelling, there are 4 cases:

case 1: $I_k = q_0$, $f=S$, and $g \neq S$. By inspection o
definition of the initial state q_0, there exis
marked production of the form $(F(\vec{x}) \rightarrow t, v0) \in q_0$
$v \in const(t)$. By inspection of the definition o
spelling, either

a) $I_k \in \delta(q_0, g)$ where $r(g)=0$

b) there exists states $k_1, \ldots, k_m \in K$ where
and $I_k \in \delta((I_{k_1}, \ldots, I_{k_m}), g)$

By inspection of the construction of the trans
δ, there can not be a marked production of the
$(F(\vec{x}) \rightarrow t, v0) \in I_k$. But this is a contradiction.

case 2: $r(f)=m>0$ and $r(g)=n>0$. From k spelli
there exists states $k_1, \ldots, k_m \in K$ such that
$I_k \in \delta((I_{k_1}, \ldots, I_{k_m}), f)$. From k spelling g, the
states $k_1', \ldots, k_n' \in K$ such that $I_k \in \delta((I_{k_1'}, \ldots, I_{k_n'}$
inspection of the construction of state I_k, it
the case that closure($\{(H(\vec{x}) \rightarrow t, u) \mid t(u)=f$,
for all i, $1 \leq i \leq m$, $(H(\vec{x}) \rightarrow t, ui) \in I_{k_i}\})$ =
closure($\{(H(\vec{x}) \rightarrow t, u) \mid t(u)=g$, for all i, $1 \leq i \leq$

$\vec{x})\text{->}t,ui)\in I_{k_i},\})$. But then it must be the case th which is a contradiction.

e 3: $r(f)m>0$ and $r(g)=0$. From k spelling f, the sts states $k_1,\ldots,k_m \in K$ such that $\delta((I_{k_1},\ldots,I_{k_m}),f)$. From k spelling g, $I_k \in \delta(q_0,g$ inspection of the construction of state I_k, it mu the case that closure($\{(H(\vec{x})\text{->}t,u) \mid t(u)=f$, all i, $1\leq i\leq m$, $(H(\vec{x})\text{->}t,ui)\in I_{k_i}\}) =$ sure($\{(H(\vec{x})\text{->}t,u) \mid t(u)=g$, $(H(\vec{x})\text{->}t,u0)\in q_0\}$). B n it must be the case that f=g which is a tradiction.

e 4: $r(f)=r(g)=0$ and $I_k \neq q_0$. From k spelling f, $\delta(q_0,f)$ while from k spelling g, $I_k \in \delta(q_0,g)$. Hen must be the case that closure($\{(H(\vec{x})\text{->}t,u) \mid t(u)$ $\vec{x})\text{->}t,u0)\in q_0\})$ = closure($\{(H(\vec{x})\text{->}t,u) \mid t(u)=g$, $\vec{x})\text{->}t,u0)\in q_0\})$. But then it must be the case tha which is a contradiction.

refore the relation spelling is a total function.

Lemma 6.3.5 (below) shows that the spelling of e stack will derive the portion of the input tree nned by its corresponding read-head.

<u>**Lemma**</u> <u>**6.3.5:**</u> Given any tree grammar $G=(\Phi,\Sigma,P,S)$, characteristic automaton $CG=(\Phi\vee\Sigma,C,\delta,q_0,F)$ where $C=\{I_1,\ldots,I_c\}$, and its BUTLR(0) parser $M=(G,K,\underline{shift},\underline{reduce},\underline{goto},\underline{start})$ where $K=\{1,\ldots,c\}$ $n\geq 1$; any $b\in 2^{N^* \times T_K}$; any three trees $t_1,t_2,t_3\in T$ that $t_1=t_3[u<-t_2]$; if $(\{(uw0,\underline{start}) \mid w\in leaf(t_2)\}\vee b,t_1) \vdash_d^n (\{(u,\alpha)\}\vee b$ then

i) $s \xrightarrow[0I]{*} t_2$ where $s=\{(w,\underline{spelling}(k)) \mid (w,k$

ii) if $\alpha(\varepsilon)\neq\underline{start}$, then for all $w\in dom(\alpha)$, $(F(\vec{x})->t,v)\in I_{\alpha(w)}$, then for all $vz\in dom($ $(F(\vec{x})->t,vz)\in I_{\alpha(wz)}$ and if $vz\notin var(t)$, t $\underline{spelling}(\alpha(wz))=t(vz)$

<u>**Proof:**</u> By induction on n.

<u>base</u> <u>case:</u> $n=1$. Hence $(\{(u0,\underline{start})\}\vee b,t_1) \vdash_d$ $(\{(u,k)\}\vee b,t_1)$ where $t_2(\varepsilon)=a\in\bar{\Sigma}$ such that $r(a)=0$ a $k=\underline{shift}(\underline{start},a)$. By the definition of <u>shift</u>, $I_k\in\delta(q_0,a)$. Hence $s=\{(\varepsilon,\underline{spelling}(k))\}=a$ and a $\xrightarrow[0I]{*}$ By the construction of the states in the characte automaton, all marked productions in I_k are in on the following two forms:

i) $(F(\vec{x}) \to t, v)$ where $t(v)=a$

ii) $(F(\vec{x}) \to t, v)$ where $v \in var(t)$ and $t(v)=x_i$ for some i, $1 \le i \le r(F)$

n both cases, $\{\varepsilon\}=\{w \mid vw \in dom(t)\}$. Hence, for all $v \in dom(t)$, clearly $(F(\vec{x}) \to t, v) \in I_{k(w)}$. Furthermore he first case, clearly $t(v)=$spelling$(k(\varepsilon))$. Hence oth conditions of the lemma are met.

nductive step; $id_1 \vdash_d^n id_2 \vdash id_3$ where $d_1=(\{(uw0,$ start$) \mid w \in leaf(t_1)\} \vee b, t_1)$ and $d_3=(\{(u,k(s_1',\ldots,s_m'))\} \vee b, t_1)$ where $r(k)=m$. Depen n the last computation performed, there are three ases:

ase 1: $id_2=(\{(ui,s_i') \mid 1 \le i \le m\} \vee b, t_1)$ where =shift$(k_1,\ldots,k_m)$, $f)$, $t_2(\varepsilon)=f \in \overline{\Sigma}$, $r(f)=m$, and for , $1 \le i \le m$, $s_i'(\varepsilon)=k_i$. Clearly, from lemma 6.3.2, fo , $1 \le i \le m$, $(\{(ujw0,$ start$) \mid i \le j \le m$, $jw \in leaf(t_2)\} \vee$ $(uj,s_j') \mid 1 \le j < i\} \vee b, t_1) \vdash_d^{n_i} (\{(ujw0,$ start$) \mid i<j$ $w \in leaf(t_2)\} \vee \{(uj,s_j') \mid 1 \le j \le i\} \vee b, t_1)$ where $0<n_i$ ence, by induction, for all i, $1 \le i \le m$, $s_i \overset{*}{\underset{OI}{\Rightarrow}} t_2/i$ here $s_i=\{(w,$ spelling$(j)) \mid (w,j) \in s_i'\}$, and for all $\in dom(s_i')$, if $(F(\vec{x}) \to t, v) \in I_{s_i'(w)}$, then for all $z \in dom(t)$, $(F(\vec{x}) \to t, vz) \in I_{s_i'(wz)}$ and if $vz \notin var(t)$, pelling$(s_i'(wz))=t(vz)$. By the definition of shif

$I_k \in \delta((I_{k_1},\ldots,I_{k_m}),f)$. Clearly $s = f(s_1,\ldots,s$
$\{(w,\underline{spelling}(j)) \mid (w,j) \in k(s'_1,\ldots,s'_m)\}$ and $s \overset{=}{_{O}}$
By construction of the states in the character
automaton CG, all marked productions in I_k are
of the following two forms:

i) $(F(\vec{x}) \text{->} t,v)$ where $t(v)=f$

ii) $(F(\vec{x}) \text{->} t,v)$ where $v \in var(t)$ and $t(v)=x$
some i, $1 \le i \le r(F)$

In the first case, clearly
$(F(\vec{x}) \text{->} t,v) \in I_{k(s'_1,\ldots,s'_m)(\varepsilon)}$. By the construc
the state $I_k \in C$, I_k=closure($\{(H(\vec{x}) \text{->} t'',w) \mid t''($
all i, $1 \le i \le m$, $(H(\vec{x}) \text{->} t'',vi) \in I_{k_i}\}$). But then,
previous induction, for all $vz \in dom(t)$,
$(F(\vec{x}) \text{->} t,vz) \in I_{k(s'_1,\ldots,s'_m)(z)}$ and
$\underline{spelling}(k(s'_1,\ldots,s'_m)(z))=t(vz)$. In the secon
since $t(v)=x_i$, it must be the case that for al
$vw \in dom(t)$, $(F(\vec{x}) \text{-=>} t,vw) \in I_{k(s'_1,\ldots,s'_m)(w)}$. He
conditions of the lemma are met.

<u>case 2:</u> $id_2=(\{(u,\beta(s'_1,\ldots,s'_m))\} \vee b,t_1)$ where
$G(\vec{x}) \text{->} t'' \in \underline{reduce}(\beta(s'_1,\ldots,s'_m)(\varepsilon))$, $r(G)=m$,
$k=\underline{goto}((k_1,\ldots,k_m),G)$ where $k_i=s'_i(\varepsilon)$ for all i
and $\beta \in skeleton(t'')$. By induction, $s' \overset{*}{\underset{OI}{\Longrightarrow}} t_2$ w
$s'=\{(w,\underline{spelling}(j)) \mid (w,j) \in \beta(s'_1,\ldots,s'_m)\}$, and

$\in dom(\beta(s_1',\ldots,s_m'))$, if $(F(\vec{x})\text{->}t,v)\in I_{\beta(s_1',\ldots,s_m')(}$
hen for all $vz\in dom(t)$, $(F(\vec{x})\text{->}t,vz)\in I_{\beta(s_1',\ldots,s_m')}$
nd if $vz\notin var(t)$, then
$\underline{\text{pelling}}(\beta(s_1',\ldots,s_m')(wz))=t(vz)$. Let
$=\{(w,\underline{\text{spelling}}(j)) \mid (w,j)\in s_i'\}$. Then
$=\{(w,\underline{\text{spelling}}(j)) \mid (w,j)\in k(s_1',\ldots,s_m')\}=G(s_1,\ldots,$
learly $(G(\vec{x})\text{->}t'',\varepsilon)\in I_{\beta(s_1',\ldots,s_m')(\varepsilon)}$ since a redu
as performed. But then, since for all $w\in dom(t'')$,
$G(\vec{x})\text{->}t'',w)\in I_{\beta(s_1',\ldots,s_m')(w))}$ and
$\underline{\text{pelling}}(\beta(s_1',\ldots,s_m')(w))=t''(w)$, $s'=t''$. Therefore
$(s_1,\ldots,s_m) \overset{}{\underset{OI}{\Longrightarrow}} t''(s_1,\ldots,s_m) \overset{*}{\underset{OI}{\Longrightarrow}} t_2$. By the
onstruction of the states of the characteristic
utomaton CG, all marked productions in I_k are in
f the following two forms:

i) $(F(\vec{x})\text{->}t,v)$ where $t(v)=G$

ii) $(F(\vec{x})\text{->}t,v)$ where $v\in var(t)$ and $t(v)=x_i$ fo
some i, $1\leq i\leq r(F)$

n the first case, clearly
$F(\vec{x})\text{->}t,v)\in I_{k(s_1',\ldots,s_m')(\varepsilon)}$. By the construction
he state $I_k\in C$, i_k=closure($\{(H(\vec{x})\text{->}t',v) \mid t'(v)=G$
ll i, $1\leq i\leq m$, $(H(\vec{x})\text{->}t',vi)\in I_{k_i}\}$). But then, for
$z\in dom(t)$, $(F(\vec{x})\text{->}t,vz)\in I_{k(s_1',\ldots,s_m')(z)}$ and
$\underline{\text{pelling}}(k(s_1',\ldots,s_m')(z))=t(vz)$. In the second ca
ince $t(v)=x_i$, it must be the case that for all

vw∈dom(t), $(F(\vec{x})\text{->}t,vw) \in I_{k(s_1',\ldots,s_m')(w)}$. Hence conditions of the lemma are met.

<u>case 3:</u> u=ε, $id_3=(\{(\varepsilon,\underline{start})\},t_1)$, and $id_2=(\{(\varepsilon,\beta)\},t_1)$ where S->t∈<u>reduce</u>(β(ε)) and β∈skeleton(t). By induction, $s' \overset{*}{\underset{OI}{\Longrightarrow}} t_2$ where s'={(w,<u>spelling</u>(j)) | (w,j)∈β}, and for all w∈dom if $(F(\vec{x})\text{->}t',v) \in I_{\beta(w)}$, then for all vz∈dom(t'), $(F(\vec{x})\text{->}t',vz) \in I_{\beta(wz)}$ and if vz∉var(t'), then <u>spelling</u>(β(wz))=t(vz). Clearly, $(S\text{->}t,\varepsilon) \in I_{\beta(\varepsilon)}$ s reduction was performed. But then, since for all w∈dom(t), $(S\text{->}t,w) \in I_{\beta(w)}$ and <u>spelling</u>(β(w))=t(w), Since, by definition, <u>spelling</u>(<u>start</u>)=S and S->t∈ clearly $S \underset{OI}{\Longrightarrow} t = s' \overset{*}{\underset{OI}{\Longrightarrow}} t_2$ meeting the condition the lemma.

Having provided the above proofs, the follow theorem puts these results together by showing th tree language accepted by a BUTLR(0) parser is identical to the tree language generated by its corresponding tree grammar.

<u>Theorem 6.3.1:</u> Given a tree grammar $G=(\Phi,\Sigma,P,S)$, characteristic automaton $CG=(\Sigma \vee \Phi, C, \delta, q_0, F)$, and i BUTLR(0) parser M=(G,K,<u>shift</u>,<u>reduce</u>,<u>goto</u>,<u>start</u>),

G) - N(M).

oof: By the definition of a tree language generat a tree grammar G, L(G) • {t€T-r | S $\overset{*}{\underset{OI}{==>}}$ t}. By t finition of the tree language accepted by a BUTLB rser M, N(M) - {t€T7r | ({(uO,start) I u€leaf(t)}, ({(S,start)},t)} . Let t€L(G) be any tree in L((nee S $\overset{OI}{=>}$ *t.* By the definition of an 01 derivati ere exists a production of the form S->s€P such t $\overset{OI}{=>}$ s ?> t • By lemma 6.3.3, (uO,start) | u€leaf(t)},t) h^{*} ({(6,s')},t) where €skeleton(s) and for all w€s, (S->s,w)€I , v. I en, since S->s€jreduce(s'(6)), ({(6,s')},t) hj (6,£t£££)},t). Hence t€N(M) and L(G) £ N(M). Or her hand, assume that t€N(M) is any tree in N(M)« nee ({(uO,start) | u€leaf(t){,t) h_d ({(6,start)}, lemma 6.3.5, S $\overset{OI}{=>}$ t. Hence t€L(G) and N(M) C I erefore L(G)»N(M).

4 **Conjectures On Determinism**

Like a LR(0) parser generator, the BUTLR(O) pi nerator does not necessarily guarantee to produce 11 defined BUTLR(O) parser. It is conjectured b thor that the BUTLR(O) parser generator will proc well defined BUTLR(O) parser if the given tree grammar is a BUOI(0) grammar. A tree grammar $G=(\Phi,\Sigma,P,S)$ is considered BUOI(0) if and only i

1) G is conservative and reduced.

2) For any two derivations $S \underset{OI}{\Longrightarrow}^{*}$ $t_1[u \leftarrow F_1(s_1,\ldots,s_m)] \underset{OI}{\Longrightarrow} t_1[u \leftarrow t(s_1,\ldots$ and $S \underset{OI}{\Longrightarrow}^{*} t_1'[u' \leftarrow F_1'(s_1',\ldots,s_m')] \underset{OI}{\Longrightarrow}$ $t_1'[u' \leftarrow t'(s_1',\ldots,s_m')]$, if there exists $v \in dom(t')$ such that

$t'(s_1',\ldots,s_m')/v=t(s_1,\ldots,s_m)$, then $v=\varepsilon$ $F_1=F_1'$, $s_i=s_i'$ for all i, $1 \leq i \leq m$, and $t_1[u \leftarrow t(s_1,\ldots,s_m)]=t_1'[u' \leftarrow t'(s_1',\ldots,s$

In other words, condition (i) guards against in nondeterminism, and condition (iii) states that reduce-move must be uniquely identified by a characteristic tree.

Chapter VII

THE MACRO LANGUAGES - AN APPLICATION

The preceeding chapters presented a new model o: e pushdown automata and a construction method to .d a deterministic parser for a subclass of the :ext-free tree languages. This chapter investiga :nteresting application: a parsing technique for ;s of string languages more general than the :ext-free string languages. That is, this chapte .oits the fact that the class of string languages ained as sets of yields of all trees in a :ext-free tree language is the class of (OI) :o-languages, which is identical to the class of exed (string) languages, (see Fischer[68][69]).

Hence, by modifying the BUTLR(0) parser to acce strings instead of trees, this chapter presents parsing method using the BUTLR(0) parser to con new parser which will recognize string language class of macro languages, as well as construct deterministic parsers for a subclass of the mac languages (which is a superset of the determini context-free string languages).

The method used to build a parser for a ma language is as follows: First, the macro gramm converted to a tree grammar G_2. Then, using th BUTLR(0) construction method, a BUTLR(0) parser built to accept the tree language generated by Finally, the constructed BUTLR(0) parsing table used to define the $BUTLR_M(0)$ parser which tests string is in the macro language generated by G_1 However, instead of using tree instantaneous descriptions and the decision relation associat the BUTLR(0) parser, the $BUTLR_M(0)$ parser has i form of instantaneous descriptions and decision relation to describe moves made by the $BUTLR_M$(0 parser.

One should note that the purpose of this chapte
to provide the motivation behind the development
BUTLR(0) parser, and to lay down the groundwork
ure research in formulating parsing methods to pa
ro languages. For these reasons, this chapter do
provide proofs for any of the theorems or
jectures. Furthermore, the construction methods
jectures presented in this chapter are new ideas
author's, are based on previous experience with
ferent forms of automata and the relationships
ween these different forms of automata, and have
n completely worked out as yet. Hence portions o
s chapter may be sketchy at best.

The chapter begins with section 7.1 by presenti
ethod of using the BUTLR(0) parser construction
hod to build a parser which simulates a LR(0) par
the class of context-free string languages.
tion 7.2 introduces the definition of macro gramm
macro languages, and provides a brief review of
ic. The chapter concludes with section 7.3 by
ending the method used in section 7.1 to simulate
ted stack automaton (see Aho[68]) instead of a PI
defines the $\text{BUTLR}_M(0)$ parser which accepts strin
guages in the class of macro languages.

7.1 Simulating LR(0) Parsers Using BUTLR(0) Pa

This section presents a method of using the BUTLR(0) parser to simulate an LR(0) parser. Th concept used is to simulate strings using "flat" where concatenation is explicitly expressed. In words, it takes a context-free string grammar G_1 converts G_1 into a tree grammar G_2 such that eve sentential form of G_2 is a "flat" tree which re the corresponding sentential form in G_1. Hence strings will be explicitly incorporated into the grammar used to generate the BUTLR(0) parser and exists an isomorphism between the sentential fo generated by the string grammar G_1 and sententi generated by the tree grammar G_2.

Back in chapter 3, pushdown automata were presented. In that chapter, both the input and stack of the PDA were presented as a string of Furthermore, the stack used $\underline{|}$ as a reserved sym represent the empty stack, concatenation as the operator used to perform a push, the top of the was assumed to be the rightmost symbol in the s and the LR(0) parser simulated a rightmost deri

To lift the above notions to a tree structure, th sentation of strings must be lifted to trees. A al assumption is to "explicitly" express the tenation operator used in generating the string. xample, the string "aAbBc" could be represented b r

```
              .                     .
             / \        or         / \
            .   c                 a   .
           / \                       / \
          .   B                     A   .
         / \                           / \
        .   b                         b   .
       / \                               / \
      .   A                             B   .
     / \                                   / \
    ⊥   a                                 c   ⊥
```

⊥ represents the empty string.

Assume the method to convert string grammars to grammars uses the tree representation on the left e). Let G be the string grammar $G=(\Phi,\Sigma,P,S)$ wher A,B,C}; Σ={a,b}; and P={S->C, A->a, B->b, C->AE B}. Following the above idea, the corresponding grammar would contain the following tree ctions:

```
S -> •          A -> •          B -> •
    / \             / \             / \
   ⊥   c           ⊥   a           ⊥   b

C -> •          C -> •
    / \             / \
   •   B           •   B
  / \             / \
 ⊥   A           •   c
                / \
               ⊥   A
```

Also, the string "ab" is derived using th

grammar G as follows:

$$S \underset{R}{\Longrightarrow} C \underset{R}{\Longrightarrow} AB \underset{R}{\Longrightarrow} Ab \underset{R}{\Longrightarrow} ab$$

The corresponding derivation using the tr

is:

```
S ==>   •      ==>   •   ==>    •     ==>      •
  OI   / \     OI   / \  OI    / \    OI      / \
      ⊥   C        ⊥   •      ⊥   •          ⊥   •
                      / \        / \            / \
                     •   B      •   •          •   •
                    / \        / \ / \        / \ / \
                   ⊥   A      ⊥  A ⊥  b      ⊥  • ⊥  b
                                               / \
                                              ⊥   a
```

By inspecting the two derivations ab

differences between them, two problems wi

representation are visibly clear. One pr

the sentential forms generated using tree

have not maintained the property that eve

form is a "flat" tree representing the co

sentential form in the string grammar. A

is that the nonterminals label leaves and

rivation (for which the BUTLR(0) parser is based o
es not necessarily correspond to a rightmost
rivation which violates the constraints of an LR(0
rser (for example, the above form of tree producti
so allow a leftmost derivation).

One should note that both of the above problems
em from a common cause. In a string grammar a
nterminal A is a placeholder which represents a se
strings (of any length) while on the other hand,
der to maintain the "flat" structure of the gener
ees, the conversion method assumes that any strin
nerated from A will be a single terminal symbol w
ll replace the node labelled by A. Clearly, such
sumption is wrong. Hence, the conversion should
ncatenate the string occurring to the left of the
nterminal A, and the string derivable from A, unt
e structure of the string derivable from A is kno

To resolve this problem, the conversion method
dified such that the rank of every nonterminal is
ised from being a constant to having an arity of
ere the nonterminal's parameter represents the st
at will occur to the left of the nonterminal. Le
e <u>string to tree conversion</u> be accomplished using
nction lift : $(\Sigma V \Phi)^* \rightarrow T_{\Sigma V \Phi V \{\cdot\}}(X_1)$ where for a

$\alpha \in (\bar{\Sigma} \vee \bar{\Phi})^*$, lift($\alpha$) is recursively defined as f

i) lift(ε) = x where ε is the empty string

```
ii) lift(βa) = •   where βa=α and a∈Σ̄
              / \
       lift(β)   a

iii) lift(βA) = A where βA=α and A∈Φ̄
                |
             lift(β)
```

<u>**Example**</u> <u>**7.1.1:**</u> Let $G=(\bar{\Phi},\bar{\Sigma},P,S)$ where

$\bar{\Phi}$ = {S, A, B, C};

$\bar{\Sigma}$ = {a, b}; and

P = {S->C, C->AB, C->ACB, A->a, B->b}. Then,

lift(ε) = x

```
lift(ab) = •
          / \
         •   b
        / \
       x   a

lift(C) = C
          |
          x

lift(AB) = A     and
           |
           B
           |
           x

lift(aAb) = •
           / \
          A   b
          |
          •
```

Using the function lift, the <u>corresponding</u> <u>t</u>
<u>grammar</u> <u>of</u> <u>a</u> <u>string</u> <u>grammar</u> $G=(\Phi,\Sigma,P,S)$, denoted
is the tree grammar $TG_G=(\Phi',\Sigma',P',S)$ where

$\Phi' = \Phi$ where $r(S)=0$ and for all $F\in(\Phi-\{S\})$, $r($

$\Sigma' = \Sigma\vee\{\cdot,\perp\}$ where $r(\cdot)=2$, $r(\perp)=0$,

and for all $a\in\Sigma$, $r(a)=0$; and

P' is a set of productions where

i) if $S\to\alpha\in P$, then $S\to lift(\alpha)(\perp)\in P'$

ii) if $A\to\alpha\in P$ where $A\neq S$, then $A(x)\to lift(\alpha)\epsilon$

iii) nothing else

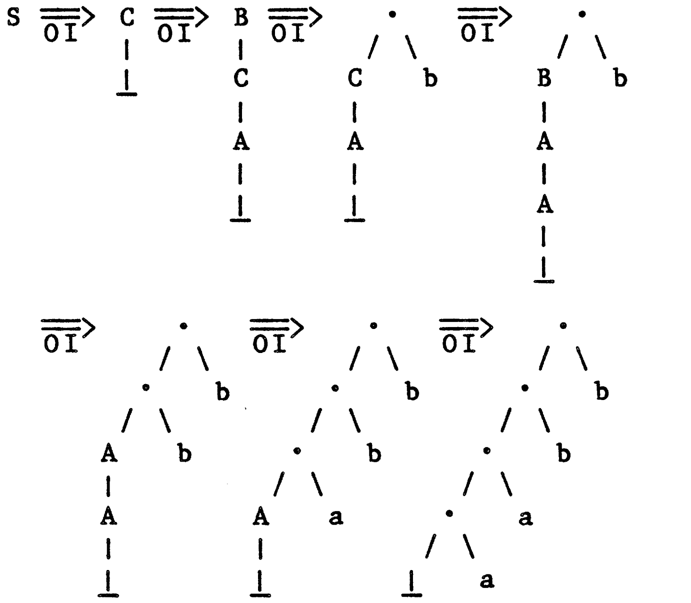
<u>**Example**</u> <u>**7.1.2:**</u> Let G be the string grammar define
example 7.1.1. The corresponding tree grammar of
the tree grammar $TG_G=(\Phi',\Sigma',P',S)$ where

```
P' = {S -> C , C -> B , C -> B ,
           |   |    |   |    |
           ⊥   x    A   x    C
                    |        |
                    x        A
                             |
                             x

      A -> •  , B -> • }.
      |   / \   |   / \
      x  x   a  x  x   b
```

Furthermore, for the derivation

$$S \underset{R}{\Longrightarrow} C \underset{R}{\Longrightarrow} ACB \underset{R}{\Longrightarrow} ACb \underset{R}{\Longrightarrow} AABb \underset{R}{\Longrightarrow}$$
$$AAbb \underset{R}{\Longrightarrow} Aabb \underset{R}{\Longrightarrow} aabb$$

the corresponding derivation in TG_G is



Note in the above example that for e

form α derived using G, the correspondin

TG_G produces the tree lift(α)($\perp$). Hence

between the string grammar and the tree g

maintained. Furthermore, since in tree p

each nonterminal occurs as a descendant o

nonterminals which occurred to the right

nonterminal in the string case, an OI der

correspond to a rightmost derivation.

One should note that the inverse operation of t
:tion lift can also be defined. Given a string
nmar $G=(\Phi,\Sigma,P,S)$ and its corresponding tree gramm
$=(\Phi',\Sigma',P',S)$, let the function
ld : $T_{\Sigma' \vee \Phi'}(X_1) \rightarrow (\Sigma \vee \Phi)^*$ be recursively defined
h that

i) $\text{yield}(x) = \varepsilon$

i) $\text{yield}(\perp) = \varepsilon$

i) $\text{yield}(a) = a$ where $a \in \Sigma$

v) $\text{yield}\left(\begin{array}{c} \cdot \\ / \ \backslash \\ t_1 \quad t_2 \end{array}\right) = \text{yield}(t_1) \cdot \text{yield}(t_2)$

v) $\text{yield}\left(\begin{array}{c} A \\ | \\ t_1 \end{array}\right) = \text{yield}(t_1) \cdot A$ where $a \in \Phi$

re t_1 and t_2 are trees in $T_{\Sigma' \vee \Phi'}(X_1)$.

mple **7.1.3:** Let G and TG_G be defined as in exampl
.2, then

$\text{yield}\left(\begin{array}{c} C \\ | \\ \perp \end{array}\right) = C$,

$\text{yield}\left(\begin{array}{c} B \\ | \\ A \\ | \\ x \end{array}\right) = AB$, and

```
yield( · ) = aAb
      / \
     A   b
     |
     •
    / \
   x   a
```

The next theorem and three lemmas show (wit
proof) the fact that the yields of trees in TG_G
string language generated by G.

__Lemma 7.1.1:__ Given any string grammar $G=(\bar{\Phi},\bar{\Sigma},P,S$
its tree grammar $TG_G=(\bar{\Phi}',\bar{\Sigma}',P',S)$, any string
$\alpha\in(\bar{\Sigma}\vee\bar{\Phi})^*$, α=yield(lift(α))=yield(lift(α)($\underline{\perp}$)

__Lemma 7.1.2:__ Given any string grammar $G=(\bar{\Phi},\bar{\Sigma},P,S$
its tree grammar $TG_G=(\bar{\Phi}',\bar{\Sigma}',P',S)$, if $S \underset{R}{\Longrightarrow}^n \alpha$,
$S \underset{OI}{\Longrightarrow}^n$ lift(α)($\underline{\perp}$) and α=yield(lift(α)($\underline{\perp}$)).

__Lemma 7.1.3:__ Given any string grammar $G=(\bar{\Phi},\bar{\Sigma},P,S$
its tree grammar $TG_G=(\bar{\Phi}',\bar{\Sigma}',P',S)$, if $S \underset{OI}{\Longrightarrow}^n t$,
$S \underset{R}{\Longrightarrow}^n$ yield(t) and t=lift(yield(t))($\underline{\perp}$).

__Theorem 7.1.1:__ Given any string grammar $G=(\bar{\Phi},\bar{\Sigma},$
its corresponding tree grammar $TG_G=(\bar{\Phi}',\bar{\Sigma}',P',S)$,
$L(G)=\{\text{yield}(t) \mid t\in L_{OI}(TG_G)\}$.

Having converted the string grammar G to the t
ammar TG_G, the $BUTLR_S(0)$ parser (the BUTLR(0) par
plied to string grammars) can be built. A $BUTLR_S$
rser is a septuple

$(G, TG_G, K, \underline{shift}, \underline{reduce}, \underline{goto}, \underline{start})$ where

$G=(\Phi, \Sigma, P, S)$ is the string grammar defining the $BUTLR_S(0)$ parser;

$TG_G=(\Phi', \Sigma', P', S)$ is the corresponding tree grammar used to define the BUTLR(0) pars tables;

K is a finite ranked alphabet of <u>parser</u> <u>states</u>;

<u>shift</u> : tuples(K) x Σ' -> K $\vee$ {<u>error</u>} is a function defining the parsing <u>shift</u> <u>table</u>;

<u>reduce</u> : K -> $2^{P'}$ is a function defining the parsing <u>reduce</u> <u>table</u>;

<u>goto</u> : tuples(K) x Φ' -> K $\vee$ {<u>error</u>} is a function defining the parsing <u>goto</u> <u>table</u>; and

<u>start</u>$\in$K is the <u>initial</u> <u>state</u>.

rthermore, the $BUTLR_S(0)$ parser is constructed us
gorithm 6.3.1. That is, let
$=(TG_G, K, \underline{shift}, \underline{reduce}, \underline{goto}, \underline{start})$ be the BUTLR(0)
rser built by algorithm 6.3.1. Then, the BUTLR(0
fines the set of states K, the initial state <u>star</u>

the three parsing functions shift, reduce, and

the $BUTLR_S(0)$ parser M, and M is deterministic

only if M′ is deterministic.

The instantaneous description of a $BUTLR_S$

parser is quite different from the instantaneou

description for the BUTLR(0) parser. The inpu

structure is a string (not a tree), the input

scanned from left to right as opposed to scann

tree from the leaves to the root, and the inte

memory is a single tree stack. An instantaneou

description of a $BUTLR_S(0)$ parser (denoted IDS

pair $(t, \alpha) \in T_K \times \overline{\Sigma}^*$ where t is the current (tr

and α is the string left to scan on the input

The initial configuration is the pair

(shift(start,|), α) where α is the string to

The decision relation $\vdash_d^s \subseteq$ IDS x IDS of a

$BUTLR_S(0)$ parser $M=(G, TG_G, K,$ shift, reduce, goto,

determines the next move made by the $BUTLR_S(0)$

M. Given two instantaneous descriptions id_1 a

$id_1 \vdash_d^s id_2$ if and only if

i) $id_1 = (t, a \cdot \alpha)$ and $id_2 = (k_2(t,k_1), \alpha)$ whe
k_1=<u>shift</u>(<u>start</u>,a) and k_2=<u>shift</u>((t(ε),k_1),·

ii) $id_1 = (\beta(t), \alpha)$ and $id_2 = (k(t), \alpha)$ where
A($\vec{x}$)->s∈<u>reduce</u>(β(t)(ε)), A≠S, β∈skeleton(s
and <u>goto</u>((t(ε)),A)=k

iii) $id_1 = (\beta, \varepsilon)$ and $id_2 =$ (<u>start</u>,ε) where
<u>reduce</u>(β(ε))={S->s} and β∈skeleton(s)

other words, in terms of an LR(0) parser, condit
) is a shift-move over the input symbol "a",
ndition (ii) is a reduce-move on the production A
ere lift(θ)=s defines the corresponding tree
oduction A(x)->s, and condition (iii) is a
duce-move on the start production S->θ causing
ceptance where lift(θ)(<u>|</u>)=s defines the correspon
art tree production S->s.

Like an LR(0) parser, acceptance of the string
ly occurs if the decision relation causes a serie
ves which converts the initial instantaneous
scription into the instantaneous description
<u>tart</u>,ε). Hence, the <u>language accepted by a</u> BUTLR
<u>rser M</u>, denoted N(M), is the set

M) = { $\alpha \in \Sigma^*$ | (<u>shift</u>(<u>start</u>,<u>|</u>),α) $\vdash_d^{s\,*}$ (<u>start</u>,ε)}
* is the transitive reflexive closure of $\vdash_d^s$.

__Example__ __7.1.4:__ Let $G=(\bar{\Phi},\Sigma,P,S)$ be a string gr

where

$\bar{\Phi} = \{S,A\}$;

$\Sigma = \{a,b\}$; and

$P = \{S\text{->}A,\ A\text{->}ab,\ A\text{->}aAb\}$;

Then $D=(G,K,$__shift__,__reduce__,__goto__,$1)$ is an LR(0)

where

$K = \{1,2,3,4,5,6\}$ and

__shift__, __reduce__, and __goto__ are defined by

the following tables:

__shift__

	a	b
1	3	
3	3	4
5		6

__reduce__

2	S->A
4	A->ab
6	A->aAb

__goto__

	A
1	2
3	5

Furthermore, $TG_G=(\bar{\Phi}',\Sigma',P',S)$ where

$\bar{\Phi}' = \bar{\Phi}$ where $r(S)=0$ and $r(A)=1$;

$\Sigma' = \Sigma \vee \{\perp,\cdot\}$ where $r(\perp)=r(a)=r(b)=0$,

and $r(\cdot)=2$; and

```
P = {S -> A , A ->    •     , A ->   •  }
          |   |      / \        |     / \
          ⊥   x     •   b       x    A   b
                   / \               |
                  x   a              •
                                    / \
                                   x   a
```

The constructed $BUTLR_S(0)$ parser M, defined b

UTLR(0) parser M′=(TG_G,K,shift,reduce,goto,1), is

=(G,TG_G,K,shift,reduce,goto,1) where

K = {1,2,3,4,5,6,7,8,9} such that

r(1)=r(2)=r(3)=r(4)=0, r(5)=r(8)=1, and

r(6)=r(7)=r(9)=2; and

shift, reduce, and goto are defined by

the following tables:

shift

	e	a	b	•
1	2	3	4	
2,3)				6
6,3)				6
6,4)				7
8,4)				9

reduce

```
  +------------+
  |    S -> A  |
5 |         |  |
  |         1  |
  +------------+
  | A ->   •   |
  | |     / \  |
7 | x    •   b |
  |     / \    |
  |    x   a   |
  +------------+
  | A ->   •   |
  | |     / \  |
  | x    A   b |
9 |      |     |
  |      •     |
  |     / \    |
  |    x   a   |
  +------------+
```

goto

	A
2	5
6	8

he language accepted by the LR(0) parser D and th

UTLR$_S$(0) parser M is the set of strings {$a^n b^n$ | r

or example, the string "aaabbb" is accepted by th

R(0) parser D as follows:

(1,aaabbb) $\vdash_d$ (13,aabbb) $\vdash_d$ (133,abbb) $\vdash_d$

(1333,bbb) $\vdash_d$ (13334,bb) $\vdash_d$ (1335,bb) $\vdash_d$

$(13356,b) \vdash_d (135,b) \vdash_d (1356,\varepsilon) \vdash_d$

$(12,\varepsilon) \vdash_d (1,\varepsilon)$

Similarly, the corresponding computation using

$BUTLR_S(0)$ parser M is as follows:

```
(2 , aaabbb) ⊢_d^s

 (6 , aabbb) ⊢_d^s
 / \
2   3

   (6 , abbb) ⊢_d^s
   / \
  6   3
 / \
2   3

     (6 , bbb) ⊢_d^s
     / \
    6   3
   / \
  6   3
 / \
2   3

       (7 , bb) ⊢_d^s
       / \
      6   4
     / \
    6   3
   / \
  6   3
 / \
2   3

   (8 , bb) ⊢_d^s
    |
    6
   / \
  6   3
 / \
```

```
  (9  , b)  ⊢s
   / \          d
 8    4
 |
 6
/ \
   3
\
 3

 , b)  ⊢s
          d

\
 3

(9 , ε)  ⊢s
/ \         d
   4


\
 3

, ε)  ⊢s
         d


, ε)
```

rom the above example, one can notice several
rities between the LR(0) parser D and the
(0) parser M. One similarity is that the string
b" is accepted by both the LR(0) parser D and the
(0) parser M by performing 10 computation moves.
r similarity is that whenever the LR(0) parser D
med a shift-move, the $BUTLR_S(0)$ parser M also
med a shift-move. The same is also true for
-moves. Also, both the LR(0) parser D and the

$BUTLR_S(0)$ parser M is deterministic. By lookin deeper similarities, one notices the similariti between the two forms of internal memory. Afte computation move, the spelling of the stack of LR(0) parser D corresponds to the yield of the of the tree stack of the $BUTLR_S(0)$ parser M. T if $(1,\alpha) \vdash_d^n (\beta,\theta)$ and $(2,\alpha) \vdash_d^{s\,n} (t,\theta)$, then $\underline{\text{spelling}}(\beta)=\text{yield}(\underline{\text{spelling}}(t))$.

It is the firm belief of the author that t results are true in general, and the following conjectures present these beliefs:

Conjecture 7.1.1: Given a string grammar G, the parser $M_1=(G,K_1,\underline{\text{shift}}_1,\underline{\text{reduce}}_1,\underline{\text{goto}}_1,\underline{\text{start}}_1)$, a $BUTLR_S(0)$ parser

$M_2=(G,TG_G,K_2,\underline{\text{shift}}_2,\underline{\text{reduce}}_2,\underline{\text{goto}}_2,\underline{\text{start}}_2)$, then

a) $(\underline{\text{start}},\alpha) \vdash_d^n (\beta,\theta)$ if and only if $(\underline{\text{shift}}_2(\underline{\text{start}},\underline{|}),\alpha) \vdash_d^{s\,n} (t,\theta)$ where $\underline{\text{spelling}}(\beta)=\text{yield}(\underline{\text{spelling}}(t))$ and $\text{lift}(\underline{\text{spelling}}(\beta))(\underline{|})=\underline{\text{spelling}}(t)$.

b) M_1 is deterministic if and only if M_2 i
deterministic

c) $L(G) = N(M_1) = N(M_2)$

Conjecture 7.1.2: Given a string grammar G, the $BUTLR_S(0)$ parser can be extended to a $BUTLR_S(k)$ (a parser with k symbols of lookahead on the inp tape). Furthermore, the conditions of conjectur apply to the comparison between the LR(k) parser the $BUTLR_S(k)$ parser.

7.2 The Macro Languages

This section presents the definition of a m grammar and how a macro (string) language is ger from a given macro grammar (see Fischer[68][69]) with most grammars, the generation process is pe via a series of derivation steps. Furthermore, tree grammars, macro grammars also have two rest forms of derivations called outside-in and insid derivations and both modes of derivation will be presented.

Informally, a macro grammar is a gene
string grammars where the notion of a macr
from programming languages. In other word
nonterminals is a ranked alphabet where no
with arity greater than zero get parameter
manner as tree grammars. The occurrences
on the right-hand side of a production cor
placeholder into which the corresponding a
the production is substituted for the occu
variable. Hence, in some sense, macro gra
quite similar to tree grammars. The diffe
that while a macro production is performin
rewrite step, the structure it is manipula
string instead of a tree.

Definition 7.2.1: A <u>macro grammar</u> is a qua
(Φ,Σ,P,S) where

Φ is a finite ranked alphabet of <u>nonterminal symbols</u>;

Σ is a finite alphabet of <u>terminal symb</u>

$S \in F$ is a designated symbol in Φ called <u>start symbol</u> where $r(S)=0$; and

P is a finite set of pairs of the form $(F(x_1,\ldots,x_m),\alpha) \in \mathbf{term}(\Phi,\Sigma)^2$ where F

Note: Each pair $(F(x_1,\ldots,x_m),\alpha)\in P$ is called a
production. Furthermore, for any production $(F(x_1,\ldots,x_m),\alpha)\in P$, if $x\in X_A$ occurs in the string then $x\in X_m$ (i.e. the only variables which can occ the right-hand side of a production are those whi occur on the left-hand side of the production).

For convenience of notation, the string $F(x_1,\ldots,x_m)$ where $r(F)=m$ will be denoted in vect form as $F(\vec{x})$. Productions will be denoted as $F(\vec{x}$ where $(F(\vec{x}),\alpha)\in P$. In general, upper case letter as F,G,H,... will be used to denote nonterminal symbols while lower case letters such as a,b,c,.. will be used to denote terminal symbols. Further greek symbols will be used to denote terms. Depe on the context, G will also be used to denote a m grammar. Finally, unless otherwise specified, on assume that $A=\max\{r(F) \mid F\in\Phi\}$.

Example 7.2.1: The macro grammar which generates strings of the form $\{a^nb^nc^n \mid n\geq 1\}$ is the macro g $G=(\Phi,\Sigma,P,S)$ where

$\Phi = \{S,F\}$ such that $r(S)=0$ and $r(F)=3$;

$\Sigma = \{a,b,c\}$; and

$P = \{S\text{->}F(a,b,c),\ F(x,y,z)\text{->}xyz,$

$F(x,y,z)\text{->}F(xa,yb,zc)\}$

A macro language is generated from a ma
by performing a series of derivation (or rew
steps. Given a macro grammar $G=(\bar{\Phi},\bar{\Sigma},P,S)$, l
<u>one-step</u> <u>derivation</u> (or rewrite) relation
$\underset{u}{\Longrightarrow} \subseteq \mathbf{term}(\bar{\Phi},\bar{\Sigma})^2$ be defined as the set of pa

$\{(\alpha\cdot F(\alpha_1,\ldots,\alpha_m)\cdot\beta, \alpha\cdot\theta[\alpha_1,\ldots,\alpha_m]\cdot\beta$

$\alpha,\alpha_1,\ldots,\alpha_m,\beta,\theta\in\mathbf{term}(\bar{\Phi},\bar{\Sigma})$, and $F(\vec{x})\text{->}\theta$

In other words, given any string $\alpha\cdot F(\alpha_1,\ldots$
string $F(\alpha_1,\ldots,\alpha_m)$ is rewritten (or repla
string $\theta[\alpha_1,\ldots,\alpha_m]$ using the production
$F(x_1,\ldots,x_m)\text{->}\theta$.

Equipped with the meaning of a one-step
derivation, one is able to define the set of
generated from a macro grammar. Given a mac
$G=(\bar{\Phi},\bar{\Sigma},P,S)$, a <u>sentential</u> <u>form</u> is any string
$\alpha\in\mathbf{term}(\bar{\Phi},\bar{\Sigma})$ such that $S \underset{u}{\Longrightarrow}^* \alpha$ where $\underset{u}{\Longrightarrow}^*$
transitive reflexive closure of $\underset{u}{\Longrightarrow}$. Furthe
<u>string</u> <u>language</u> <u>generated</u> <u>by</u> <u>a</u> <u>macro</u> <u>grammar</u>
$L(G)$, is the set of all sentential forms α
$\alpha\in\bar{\Sigma}^*$. Hence

$$L(G)=\{\alpha\in\bar{\Sigma}^* \mid S \underset{u}{\Longrightarrow}^* \alpha\}.$$

Example 7.2.2: Let G be the macro language define example 7.2.1. A sample derivation which generat string "aaabbbccc" is as follows:

$$S \underset{u}{\Longrightarrow} F(a,b,c) \underset{u}{\Longrightarrow} F(aa,bb,cc) \underset{u}{\Longrightarrow}$$
$$F(aaa,bbb,ccc) \underset{u}{\Longrightarrow} aaabbbccc$$

As mentioned earlier, the situation regardin derivation modes for macro grammars is as intrica the derivation modes for tree grammars. One-step derivations are not commutative in the sense that $\alpha_1 \underset{u}{\Longrightarrow} \alpha_2$ using $F_1(\vec{x})\text{->}\beta_1$ and $\alpha_2 \underset{u}{\Longrightarrow} \alpha_3$ using $F_2(\vec{x})\text{->}\beta_2$, it is not necessarily the case that th exists a term α'_2 such that $\alpha_1 \underset{u}{\Longrightarrow} \alpha'_2$ using $F_2($ and $\alpha'_2 \underset{u}{\Longrightarrow} \alpha_3$ using $F_1(\vec{x})\text{->}\beta_1$. To show this, c the following example:

Example 7.2.3: Let $G=(\Phi,\Sigma,P,S)$ be a macro grammar that

Φ = {S,F,G} where R(S)=0 and r(F)=r(G)=1;

Σ = {a}; and

P = {S->F(G(a)), F(x)->a, G(x)->x}.

Clearly $F(G(a)) \underset{u}{\Longrightarrow} F(a)$ using G(x)->x and F(a) = using F(x)->a. On the other hand, when the order the derivation steps is reversed, $F(G(a)) \underset{u}{\Longrightarrow} a$ u F(x)->a and it is now impossible to perform a rew using G(x)->x.

Hence, the order in which derivation steps a applied affects the resulting derived string (i.e derivation steps are not necessarily independent another). This result has been shown by Fischer[68][69]. Like tree grammars, there are t modes of derivations (besides the unrestricted ca which are commonly used and are known as inside-o (IO) or outside-in (OI) derivation modes.

An <u>IO</u> <u>one-step</u> <u>derivation</u> (denoted $\underset{IO}{\Longrightarrow}$) is a one-step derivation applied to an innermost nonte occurring in the string. In other words, the derivation step can be applied to any subterm $F(\alpha_1,\ldots,\alpha_m)$ where no nonterminals occur in any terms α_1 through α_m. More formally, the $\underset{IO}{\Longrightarrow}$ re is defined as follows:

For any two terms $\alpha_1,\alpha_2 \in \mathbf{term}(\Phi,\Sigma)$, $\alpha_1 \underset{IO}{\Longrightarrow}$ only if $\alpha_1 \underset{u}{\Longrightarrow} \alpha_2$ and

i) $\alpha_1 = \alpha F(\beta_1,\ldots,\beta_m)\theta$

ii) $\alpha_2 = \alpha\delta[\beta_1,\ldots,\beta_m]\theta$

.i) $F(\vec{x}) \to \delta \in P$ where $r(F)=m$

.v) for all i, $1 \leq i \leq m$, $\beta_i \in (\Sigma \vee X_A)^*$

:e that conditions (i) through (iii) are just the
ıditions of a one-step derivation while condition
ı) is the added condition of an IO derivation.

Similarly, an <u>OI one-step derivation</u> (denoted
a one-step derivation applied to a top-level
ıterminal in a term. In other words, it can be
ılied to any nonterminal which is not embedded wi
ıther nonterminal. More formally, the relation $\overline{\overline{0}}$
defined as follows:

for any two terms $\alpha_1, \alpha_2 \in term(\bar{\Phi},\bar{\Sigma})$, $\alpha_1 \underset{OI}{\Longrightarrow} \alpha_2$
only if $\alpha_1 \underset{u}{\Longrightarrow} \alpha_2$ and

i) $\alpha_1 = \alpha F(\beta_1,\ldots,\beta_m)\theta$

ii) $\alpha_2 = \alpha \delta[\beta_1,\ldots,\beta_m]\theta$

ii) $F(\vec{x}) \to \delta \in P$ where $r(F)=m$

iv) $\alpha, \theta \in term(\bar{\Phi},\bar{\Sigma})$

ain, as in an IO one-step derivation, conditions
rough (iii) are just the conditions for a one-ste
rivation while condition (iv) is the added condit
r an OI derivation.

To clarify the difference between unrestri

IO, and OI derivations (i.e. $\overset{}{\underset{u}{\Longrightarrow}}$, $\underset{IO}{\Longrightarrow}$, and $\underset{OI}{\Longrightarrow}$

consider the following example:

Example 7.2.4: Let $G=(\Phi,\Sigma,P,S)$ be a macro gramm

that

Φ = {S,F,G} where r(S)=0 and r(F)=r(G)=1;

Σ = {a}; and

P = {S->F(G(a)), F(x)->xx, G(x)->xx, G(x)->x

The set of all possible IO derivations is as fo

$S \underset{IO}{\Longrightarrow} F(G(a)) \underset{IO}{\Longrightarrow} F(aa) \underset{IO}{\Longrightarrow} aaaa$

$S \underset{IO}{\Longrightarrow} F(G(a)) \underset{IO}{\Longrightarrow} F(a) \underset{IO}{\Longrightarrow} aa$

On the other hand, the set of all possible OI

derivations is as follows:

$S \underset{OI}{\Longrightarrow} F(G(a)) \underset{OI}{\Longrightarrow} G(a)G(a) \underset{OI}{\Longrightarrow} aaG(a) \underset{OI}{\Longrightarrow} aa$

$S \underset{OI}{\Longrightarrow} F(G(a)) \underset{OI}{\Longrightarrow} G(a)G(a) \underset{OI}{\Longrightarrow} aaG(a) \underset{OI}{\Longrightarrow} aa$

$S \underset{OI}{\Longrightarrow} F(G(a)) \underset{OI}{\Longrightarrow} G(a)G(a) \underset{OI}{\Longrightarrow} aG(a) \underset{OI}{\Longrightarrow} aaa$

$S \underset{OI}{\Longrightarrow} F(G(a)) \underset{OI}{\Longrightarrow} G(a)G(a) \underset{OI}{\Longrightarrow} aG(a) \underset{OI}{\Longrightarrow} aa$

$S \underset{OI}{\Longrightarrow} F(G(a)) \underset{OI}{\Longrightarrow} G(a)G(a) \underset{OI}{\Longrightarrow} G(a)aa \underset{OI}{\Longrightarrow} aa$

$S \underset{OI}{\Longrightarrow} F(G(a)) \underset{OI}{\Longrightarrow} G(a)G(a) \underset{OI}{\Longrightarrow} G(a)aa \underset{OI}{\Longrightarrow} aa$

$S \underset{OI}{\Longrightarrow} F(G(a)) \underset{OI}{\Longrightarrow} G(a)G(a) \underset{OI}{\Longrightarrow} G(a)a \underset{OI}{\Longrightarrow} aaa$

$S \underset{OI}{\Longrightarrow} F(G(a)) \underset{OI}{\Longrightarrow} G(a)G(a) \underset{OI}{\Longrightarrow} G(a)a \underset{OI}{\Longrightarrow} aa$

Also, the set of all possible (unrestricted)

derivations is as follows:

$S \underset{u}{\Longrightarrow} F(G(a)) \underset{u}{\Longrightarrow} G(a)G(a) \underset{u}{\Longrightarrow} aaG(a) \underset{u}{\Longrightarrow} aa$

$S \underset{u}{\Longrightarrow} F(G(a)) \underset{u}{\Longrightarrow} G(a)G(a) \underset{u}{\Longrightarrow} aaG(a) \underset{u}{\Longrightarrow} aaa$

$S \underset{u}{\Longrightarrow} F(G(a)) \underset{u}{\Longrightarrow} G(a)G(a) \underset{u}{\Longrightarrow} aG(a) \underset{u}{\Longrightarrow} aaa$

$S \underset{u}{\Longrightarrow} F(G(a)) \underset{u}{\Longrightarrow} G(a)G(a) \underset{u}{\Longrightarrow} aG(a) \underset{u}{\Longrightarrow} aa$

$S \underset{u}{\Longrightarrow} F(G(a)) \underset{u}{\Longrightarrow} G(a)G(a) \underset{u}{\Longrightarrow} G(a)aa \underset{u}{\Longrightarrow} aaaa$

$S \underset{u}{\Longrightarrow} F(G(a)) \underset{u}{\Longrightarrow} G(a)G(a) \underset{u}{\Longrightarrow} G(a)aa \underset{u}{\Longrightarrow} aaa$

$S \underset{u}{\Longrightarrow} F(G(a)) \underset{u}{\Longrightarrow} G(a)G(a) \underset{u}{\Longrightarrow} G(a)a \underset{u}{\Longrightarrow} aaa$

$S \underset{u}{\Longrightarrow} F(G(a)) \underset{u}{\Longrightarrow} G(a)G(a) \underset{u}{\Longrightarrow} G(a)a \underset{u}{\Longrightarrow} aa$

$S \underset{u}{\Longrightarrow} F(G(a)) \underset{u}{\Longrightarrow} F(aa) \underset{u}{\Longrightarrow} aaaa$

$S \underset{u}{\Longrightarrow} F(G(a)) \underset{u}{\Longrightarrow} F(a) \underset{u}{\Longrightarrow} aa$

Note that in the above example that using an
erivation mode the language generated is {aa,aaa
hile using either an OI or unrestricted derivatic
ode the language generated is {aa,aaa,aaaa}. As
ight expect, it turns out that the results about
O, OI, and unrestricted modes of derivation for t
anguages are also true for macro grammars. Howev
efore stating these results the definition of a m
anguage must be extended to allow the derivation
o be specified.

For notational convenience, the transitive
losures of the different derivation modes are def
s follows. The transitive closure of $\underset{u}{\Longrightarrow}$, $\underset{\overline{IO}}{\Longrightarrow}$, a
$\underset{\overline{I}}{\Longrightarrow}$ are denoted as $\underset{u}{\Longrightarrow}^{+}$, $\underset{\overline{IO}}{\Longrightarrow}^{+}$, and $\underset{\overline{OI}}{\Longrightarrow}^{+}$ respective

while the transitive reflexive closures of $\underset{u}{\Longrightarrow}$
and $\underset{OI}{\Longrightarrow}$ are denoted as $\underset{u}{\Longrightarrow}^*$, $\underset{IO}{\Longrightarrow}^*$, and $\underset{OI}{\Longrightarrow}^*$
respectively.

To extend the notion of a macro language either an IO, OI, or unrestricted derivation m must also generalize the definition of sentent forms. Given a macro grammar $G=(\Phi,\Sigma,P,S)$ and derivation relation $\underset{M}{\Longrightarrow}$ where $M\in\{IO,OI,u\}$, a s <u>form</u> is any term α such that $S \underset{M}{\Longrightarrow}^* \alpha$. Furt the <u>string language generated</u> by G using $\underset{M}{\Longrightarrow}$, $L_M(G)$, is the set $\{\alpha\in\Sigma^* \mid S \underset{M}{\Longrightarrow}^* \alpha\}$.

Having generalized these definitions, th following result of Fischer[68][69] is presen without proof:

<u>Theorem 7.2.1:</u> Given a macro grammar $G=(\Phi,\Sigma,P$ macro languages generated by the three differ of derivation are related as follows:

$$L_{IO}(G) \subseteq L_{OI}(G) = L_u(G)$$

<u>Note:</u> The remainder of this chapter will only OI derivations.

The object of this chapter is to apply the
ITLR(0) parser to string languages and attempt to
roduce as much determinism as is possible. Follo
his notion, the methods used to make LR(0) parser
eterministic should also be applied to macro
nguages. To this end, the notion of a rightmost
erivation must also be introduced. Given a macro
rammar $G=(\bar{\Phi},\bar{\Sigma},P,S)$, the <u>one-step</u> <u>rightmost</u> <u>deriva</u>
elation $\underset{R}{\Longrightarrow} \subseteq \mathrm{term}(\bar{\Phi},\bar{\Sigma})^2$ is defined as follows:
For any two terms $\alpha_1,\alpha_2 \in \mathrm{term}(\bar{\Phi},\bar{\Sigma})$, $\alpha_1 \underset{R}{\Longrightarrow} \alpha$
only if $\alpha_1 \underset{OI}{\Longrightarrow} \alpha_2$ and

i) $\alpha_1 = \alpha F(\beta_1,\ldots,\beta_m)\theta$

ii) $\alpha_2 = \alpha\delta[\beta_1,\ldots,\beta_m]\theta$

iii) $F(\vec{x})\rightarrow\delta \in P$ where $r(F)=m$

iv) $\theta \in (\bar{\Sigma} \vee X_A)^*$

n other words, $\underset{R}{\Longrightarrow}$ is the one-step derivation app
o the rightmost top-level nonterminal. Furthermo
et $\underset{R}{\Longrightarrow}^+$ and $\underset{R}{\Longrightarrow}^*$ denote the transitive and transi
eflexive closures of $\underset{R}{\Longrightarrow}$ respectively.

<u>Example</u> <u>7.2.5:</u> Let G be the macro grammar defined example 7.2.4. The set of all possible rightmost derivations is as follows:

$$S \underset{R}{\Longrightarrow} F(G(a)) \underset{R}{\Longrightarrow} G(a)G(a) \underset{R}{\Longrightarrow} G(a)aa \underset{R}{\Longrightarrow} aaaa$$

$$S \underset{R}{\Longrightarrow} F(G(a)) \underset{R}{\Longrightarrow} G(a)G(a) \underset{R}{\Longrightarrow} G(a)aa \underset{R}{\Longrightarrow} aaa$$

$$S \underset{R}{\Longrightarrow} F(G(a)) \underset{R}{\Longrightarrow} G(a)G(a) \underset{R}{\Longrightarrow} G(a)a \underset{R}{\Longrightarrow} aaa$$

$$S \underset{R}{\Longrightarrow} F(G(a)) \underset{R}{\Longrightarrow} G(a)G(a) \underset{R}{\Longrightarrow} G(a)a \underset{R}{\Longrightarrow} a$$

As with IO and OI derivation modes, in order extend the notion of a macro language under a rig derivation, the definition of sentential forms mu again be generalized. Given a macro grammar $G=(\Phi,\Sigma,P,S)$ and the derivation relation $\underset{R}{\Longrightarrow}$, a <u>sentential</u> <u>form</u> is any term α such that $S \underset{R}{\Longrightarrow}^{*} \alpha$ Furthermore, the <u>string</u> <u>language</u> generated by G u $\underset{R}{\Longrightarrow}$, denoted $L^{R}_{OI}(G)$, is the set $\{\alpha \in \Sigma^{*} \mid S \underset{R}{\Longrightarrow}^{*} \alpha$

Having generalized the above definitions, th following result is conjectured:

<u>Conjecture</u> <u>7.2.1:</u> Given a macro grammar G, $L^{R}_{OI}(G) = L_{OI}(G) = L(G)$.

Parsing The Macro Languages

This section presents a new type of parser to ognize string languages in the class of OI macro guages, the $BUTLR_M(0)$ parser (the BUTLR(0) parser lied to macro languages). The $BUTLR_M(0)$ parser i structed using the construction method for the LR(0) parser and is a generalization of the $LR_S(0)$ parser presented in section 7.1.

The method used to build the $BUTLR_M(0)$ parser i follows: First, the macro grammar G_1 is converte a tree grammar G_2 using a generalization of the ction "lift" defined in section 7.1. Then, using BUTLR(0) construction method, a BUTLR(0) parser built to accept the tree language generated by G_2 ally, the parsing tables of the BUTLR(0) parser M used to define the $BUTLR_M(0)$ parser M where the ernal memory simulates a nested stack automaton sented by Aho[69].

A nested stack automaton is a parser invented b to parse the class of indexed languages (which i ntical to the class of OI macro languages, see cher[68][69]). The parser is a nondeterministic -down parsing method where the moves of the neste ck automaton simulate the derivation which produc the input string. While the nested stack automa quite interesting in itself, the important conce in this thesis is its form of internal memory, t nested stack.

A nested stack is a recursively defined ob based on the notion of a stack. Like a stack, t are only two operators which update the stack. operators are the push (adds an element to the t the stack) and the pop (deletes an element from of the stack). However, unlike the typical defi of a stack, an element on the stack can either b stack symbol or a nested stack (and hence, a recursively defined object).

The way in which the nested stack is used i nested stack automaton to parse macro languages simulate a PDA whenever possible. The top-level is used to parse the top-level strings (strings embedded by nonterminals with arity greater than Hence, whenever the macro grammar is also a stri grammar, the nested stack is just a stack and th nested stack automaton simulates a PDA. Howeve nonterminal occurs in the top-level string, and nonzero arity, a nested stack is created for ea its parameters. Then, each of the parameters ar

reated like a top-level string and are parsed usi
ne same method as with the top-level string (and
ence, uses the recursive nature of the nested sta

To use the nested stack in the construction o
JTLR$_M$(0) parser, the function "lift" has to be
eneralized such that a macro grammar is converted
ree grammar which will simulate a nested stack in
a stack (as was done in section 7.1). In other
ords, the arity of each nonterminal will be raise
ne where the added parameter of the nonterminal
epresents the string that will occur to the left
ne nonterminal. A stack will be used to parse th
op-level string under the assumptions used by an
arser (i.e. under the same assumptions used by t
JTLR$_S$(0) parser). Whenever a nonterminal with ar
ppears in the string, a nested stack will be crea
or each parameter of the nonterminal and each nes
tack will be treated like a top level string. In
ther words, each nested stack will be parsed unde
ssumptions used by a LR(0) parser. Furthermore,
ifferentiate between the top-level stack and a ne
tack two empty stack symbols will be used. The s
$\perp$" will be used as the empty stack symbol for the
op-level stack while the symbol "ε" will be used
ne empty stack symbol for all nested stacks.

Given a macro grammar $G=(\overline{\Phi},\overline{\Sigma},P,S)$, let th

function lift : $\mathrm{term}(\overline{\Phi},\overline{\Sigma}) \rightarrow T_{\overline{\Sigma}\vee\overline{\Phi}'\vee\{\cdot,\perp,\varepsilon\}}(\mathbf{X}_A$

where $\overline{\Phi}'=\overline{\Phi}$ such that for all $F\in\overline{\Phi}'$ where $F\neq S$, t

of F is one larger than its corresponding rank

and lift is recursively defined as follows:

(i) $\mathrm{lift}(\varepsilon) = x_1$ where ε is the empty string

(ii) $\mathrm{lift}(\beta\cdot x_i) = \begin{array}{ccc} & \cdot & \\ / & & \backslash \\ \mathrm{lift}(\beta) & & x_{i+1} \end{array}$ where $x_i \in \mathbf{X}_A$

(iii) $\mathrm{lift}(\beta\cdot a) = \begin{array}{ccc} & \cdot & \\ / & & \backslash \\ \mathrm{lift}(\beta) & & a \end{array}$ where $a\in\overline{\Sigma}$

(iv) $\mathrm{lift}(\beta\cdot F) = \begin{array}{c} F \\ | \\ \mathrm{lift}(\beta) \end{array}$ where $F\in\overline{\Phi}$ and $r(F)=0$

(v) $\mathrm{lift}(\beta.F(\beta_1,\ldots,\beta_m)) = \begin{array}{c} F \\ \mathrm{lift}(\beta)\ \mathrm{lift}(\beta_1)(\varepsilon)\ \ldots\ \mathrm{lift}(\end{array}$

where $F\in\overline{\Phi}$ and $r(F)=m>0$

Example 7.3.1: Let G be the string grammar def

example 7.1.1. Then,

$\mathrm{lift}(\varepsilon) = x$

$\mathrm{lift}(C) = \begin{array}{c} C \\ | \\ x \end{array}$

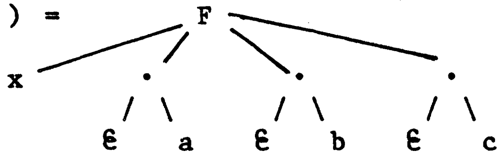
$\mathrm{lift}(AB) = \begin{array}{c} A \\ | \\ B \\ | \\ x \end{array}$ and

```
t(aAb) =   •
          / \
         A   b
         |
         •
        / \
       x   a
```

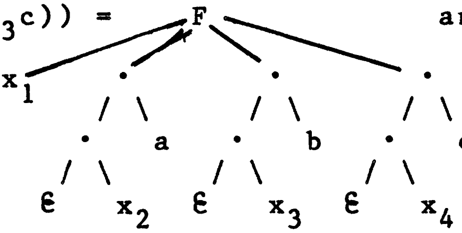
er words, the function "lift" is a generalization function "lift" presented in section 7.1 and for string grammars the resulting trees will be me.

<u>e 7.3.2:</u> Let G be the macro grammar in example Then,

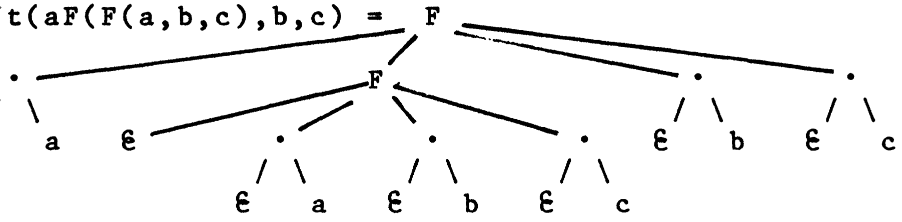
t(F(a,b,c)) =



t(F(x_1a,x_2b,x_3c)) = and



t(aF(F(a,b,c),b,c) =



Using the function lift, the method to co macro grammar to a tree grammar can be defined a macro grammar $G=(\Phi,\Sigma,P,S)$, let the <u>correspon</u> <u>grammar of the macro grammar</u> G (denoted TG_G) b tree grammar $TG_G=(\Phi',\Sigma',P',S)$ where

$\Phi' = \Phi$ where $r(S)=0$ and for all other nonterminals $F\in\Phi'$, the rank of F is one than the rank of F in Φ;

$\Sigma' = \Sigma\cup\{\cdot,\perp,\varepsilon\}$ where $r(\perp)=r(\varepsilon)=0$, $r(\cdot)=2$, and for all $a\in\Sigma$, $r(a)=0$; and

P' is a set of productions where

i) if $S\to\alpha\in P$, then $S\to lift(\alpha)(\perp)\in P'$

ii) if $A\to\alpha\in P$ where $A\neq S$ and $r(A)=0$, then $A(x)\to lift(\alpha)\in P'$

iii) if $A(x_1,\ldots,x_m)\to\alpha\in P$ where $r(F)=m>0$, t $A(x_1,\ldots,x_{m+1})\to lift(\alpha)\in P'$

iv) nothing else

<u>Example</u> <u>7.3.3:</u> Let $G=(\Phi,\Sigma,P,S)$ be the macro gr such that

$\Phi = \{S,F\}$ where $r(S)=0$ and $R(F)=3$,

$\Sigma = \{a,b,c,d\}$, and

$P = \{S \to F(ad,bd,cd),$

$F(x,y,z) \rightarrow F(axd,byd,czd)$,

$F(x,y,z) \rightarrow xdydzd\}$.

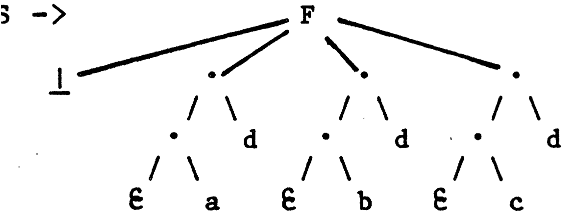
language generated by G is the set

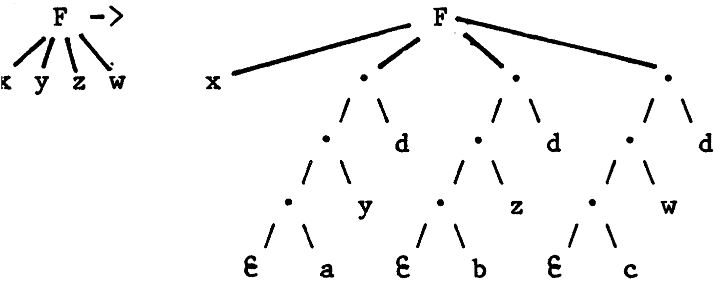
$\{a^n d^{n+1} b^n d^{n+1} c^n d^{n+1} \mid n \geq 1\}$.

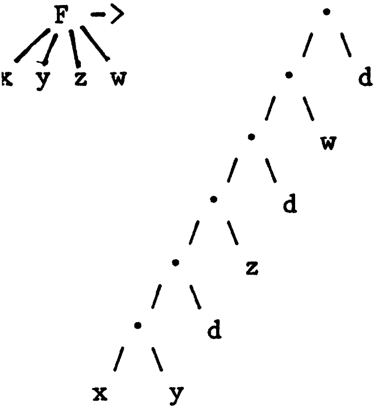
n the macro grammar G is converted to the tree

nmar TG_G, the generated tree productions are as

lows:







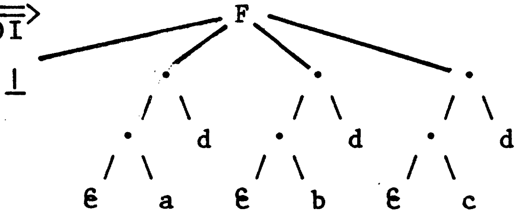
thermore, for the derivation

$S \underset{R}{\Longrightarrow} F(ad,bd,cd) \underset{R}{\Longrightarrow} F(aadd,bbdd,ccdd) \underset{R}{\Longrightarrow}$

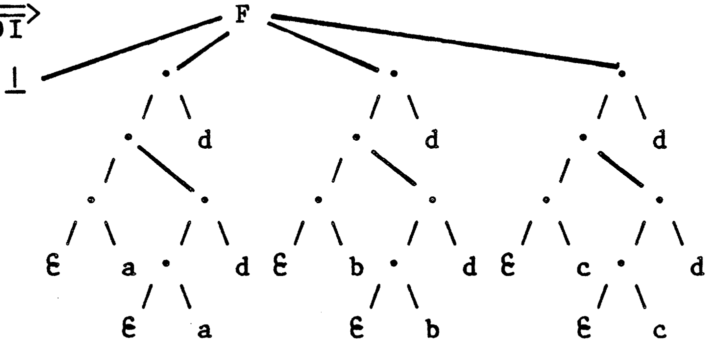
$F(aaaddd,bbbddd,cccddd) \overset{}{\underset{R}{\Longrightarrow}}$ aaadddbbbdddc

the corresponding derivation in TG_G is

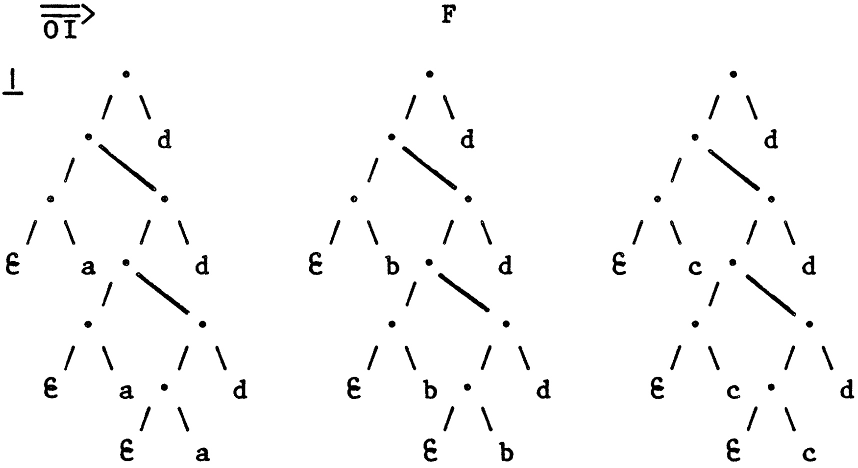
$S \underset{OI}{\Longrightarrow}$

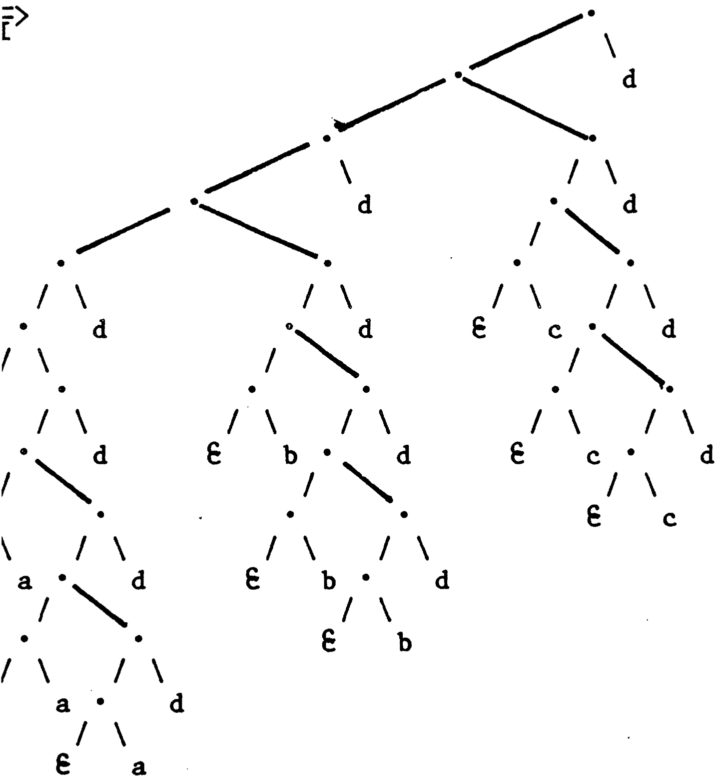


$\underset{OI}{\Longrightarrow}$



$\underset{OI}{\Longrightarrow}$





e: The tree derived by TG_G in the above derivatio

resents the nested stack $\perp \cdot L_1 \cdot d \cdot L_4 \cdot d \cdot L_7 \cdot d$ where L

ough L_9 are nested stacks as follows:

$L_1 = \varepsilon \cdot a \cdot L_2 \cdot d$

$L_2 = \varepsilon \cdot a \cdot L_3 \cdot d$

$L_3 = \varepsilon \cdot a \cdot d$

$L_4 = \varepsilon \cdot b \cdot L_5 \cdot d$

$L_5 = \varepsilon \cdot b \cdot L_6 \cdot d$

$L_6 = \varepsilon \cdot b \cdot d$

$L_7 = \varepsilon \cdot c \cdot L_8 \cdot d$

$L_8 = \varepsilon \cdot c \cdot L_9 \cdot d$

$L_9 = \varepsilon \cdot c \cdot d$

One should note that the inverse mapping of function lift can also be defined. Given a macr grammar $G=(\bar{\Phi},\bar{\Sigma},P,S)$ and its corresponding tree g $TG_G=(\bar{\Phi}',\bar{\Sigma}',P',S)$, let the function $\text{yield} : T_{\bar{\Sigma}' \vee \bar{\Phi}'}(\mathbf{X}_A) \rightarrow \mathbf{term}(\bar{\Phi},\bar{\Sigma})$, where $A=\max\{r(F) \mid F\in\bar{\Phi}'\}$, be recursively defined as fo

i) $\text{yield}(x_1) = \text{yield}(\underline{|}) = \text{yield}(\varepsilon) = \varepsilon$

ii) $\text{yield}(a) = a$ where $a\in\bar{\Sigma}$

iii) $\text{yield}(x_{i+1}) = x_i$ where $x_{i+1}\in\mathbf{X}_A$ and $i\geq 1$

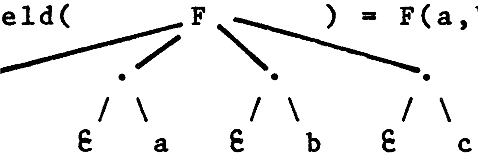
iv) $\text{yield}\left(\begin{array}{c} \cdot \\ / \quad \backslash \\ t_1 \quad t_2 \end{array}\right) = \text{yield}(t_1) \cdot \text{yield}(t_2)$

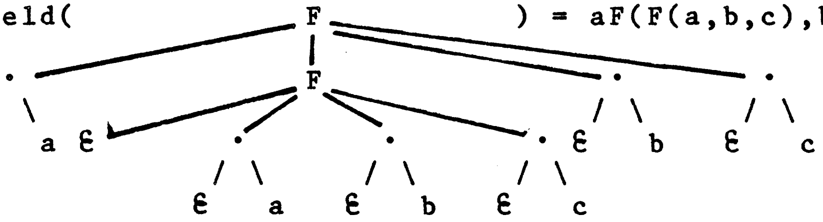
v) $\text{yield}\left(\begin{array}{c} F \\ | \\ t_1 \end{array}\right) = t_1 \cdot F$ where $F\in\bar{\Phi}'$

vi) $\text{yield}\left(\begin{array}{c} F \\ \\ t_1 \quad \cdots \quad t_{m+1} \end{array}\right) = t_1 \cdot F(\text{yield}(t_2),\ldots,\text{yiel}$

where $F\in\bar{\Phi}'$ and $t_1,\ldots,t_A \in T_{\bar{\Sigma}' \vee \bar{\Phi}'}(\mathbf{X}_A)$

ample 7.3.4: Let G and TG_G be defined as in examp
3.3, then

eld() = F(a,b,c) and

eld() = aF(F(a,b,c),b,c)

The next theorem and three lemmas state (witho
oof) that the set of yields of trees in TG_G is th
cro language generated by the macro grammar G.

mma 7.3.1: Given any macro grammar $G=(\bar{\Phi},\bar{\Sigma},P,S)$ an
s tree grammar $TG_G=(\bar{\Phi}',\bar{\Sigma}',P',S)$, any string
$\in$term$(\bar{\Phi},\bar{\Sigma})$, α=yield(lift(α)) = yield(lift(α)(

mma 7.3.2: Given any macro grammar $G=(\bar{\Phi},\bar{\Sigma},P,S)$ an
s tree grammar $TG_G=(\bar{\Phi}',\bar{\Sigma}',P',S)$, if $S \overset{n}{\underset{R}{\Longrightarrow}} \alpha$, th
$\overset{n}{\underset{OI}{\Longrightarrow}}$ lift(α)($\underline{\perp}$) and α=yield(lift(α)($\underline{\perp}$)).

Lemma 7.3.3: Given any macro grammar $G=(\Phi,\Sigma,P,$
its tree grammar $TG_G=(\Phi',\Sigma',P',S)$, if $S \underset{OI}{\Rightarrow}^n$ t
$S \underset{R}{\Rightarrow}^n$ yield(t) and t=lift(yield(t)($\perp$)).

Theorem 7.3.1: Given any macro grammar $G=(\Phi,\Sigma,$
its tree grammar $TG_G=(\Phi',\Sigma',P',S)$,
$L^R_{OI}(G) = \{\text{yield}(t) \mid t \in L_{OI}(TG_G)\}$.

Having converted the macro grammar G to t
grammar TG_G, the $BUTLR_M(0)$ parser (the BUTLR(0
applied to the macro grammars) can be built.
$BUTLR_M(0)$ parser is a septuple

$M=(G,TG_G,K,$<u>shift</u>,<u>reduce</u>,<u>goto</u>,<u>start</u>) where

- $G = (\Phi,\Sigma,P,S)$ is the macro grammar definin the BUTLR(0) parser;
- $TG_G = (\Phi',\Sigma',P',S)$ is the corresponding tree grammar used to define the BUTLR(0 tables;
- K is a finite ranked alphabet of <u>parser</u> <u>sta</u>
- <u>shift</u> : tuples(K) x Σ' -> $K \cup \{$<u>error</u>$\}$ is a function defining the parsing <u>shift</u>
- <u>reduce</u> : K -> $2^{P'}$ is a function defining the parsing <u>reduce</u> <u>table</u>;

goto : tuples(K) x Φ' -> K$\vee${error}

is a function defining the parsing goto table; and

start$\in$K is the initial state;

urthermore, the BUTLR$_M$(0) parser is constructed u lgorithm 6.3.1. Let

$'$=(TG$_G$,K,shift,reduce,goto,start) be the BUTLR(0) arser built by algorithm 6.3.1. Then, the BUTLR(arser M$'$ defines the set of states K, the initial tate start, and the three parsing functions shift educe, and goto of the BUTLR$_M$(0) parser M.

The instantaneous description of the BUTLR$_M$(0 arser is quite different from the instantaneous escriptions for the BUTLR(0) parser. The input tructure is a string (not a tree) and is scanned eft to right (as opposed to scanning the tree fro eaves to the root). Furthermore, the internal me s a string of tree stacks where each element in t tring is a tree stack representing a nested stack he tree stacks in the string are ordered in a eft-to-right order according to the relative nest f the nested stack the tree stack is representing ther words, the string of tree stacks is a list o ested stacks where the first element in the list he top-level stack and all other elements are nes stacks which have not yet been added (pushed) to the top-level stack. More formally, an instantaneous description of a $BUTLR_M(0)$ parser (denoted IDM) is a pair $(\alpha,\beta)\in T_K^* \times \Sigma^*$ where α is a string of trees and β is the remaining portion of the input string that has not been read. The initial configuration is the pair (k,β) where $k=\underline{shift}(\underline{start},\underline{|})$ and β is the string to parse.

The decision relation $\vdash_d^m \subseteq$ IDM x IDM of a parser $M=(G,TG_G,\underline{shift},\underline{reduce},\underline{goto},\underline{start})$ determines the next move made by the $BUTLR_M(0)$ parser M. Given instantaneous descriptions id_1 and id_2, $id_1 \vdash_d^m id_2$ if and only if one of the following six conditions hold:

(i) $id_1 = (\alpha\cdot t,a\cdot\beta)$ and $id_2 = (\alpha\cdot k_2(t,k_1),\beta)$ where $k_1=\underline{shift}(\underline{start},a)$ and $k_2=\underline{shift}((t(\varepsilon),k_1),\cdot)$

In other words, this condition is a shift-move applied to the innermost (rightmost) nested stack where the input symbol a is read and the corresponding state k_1 is pushed onto the innermost (rightmost) nested stack.

(ii) $id_1 = (\alpha\cdot t(t_1,\ldots,t_m),\beta)$ and
$id_2 = (\alpha\cdot k(t_1,\ldots,t_m),\beta)$ where
$F(\vec{x})\text{->}s\in\underline{reduce}(t(t_1,\ldots,t_m)(\varepsilon))$, $F\neq S$, $r(F)=m$,

skeleton(s), and $\underline{goto}((t_1(\varepsilon),\ldots,t_m(\varepsilon)),F)=k$

te that this condition is a reduce-move applied t
e innermost (rightmost) nested stack using the ma
oduction $F(\vec{x})\text{->}\theta$ where $lift(\theta)=s$ defines the
rresponding tree production $F(\vec{x})\text{->}s$.

ii) $id_1 = (\alpha, a\cdot\beta)$ and $id_2 = (\alpha\cdot k(k_1,k_2),\beta)$ wher
$=\underline{shift}(\underline{start},\varepsilon)$, $k_2=\underline{shift}(\underline{start},a)$, and
$\underline{shift}((k_1,k_2),\cdot)$

is condition creates a new one-node nested stack
ith the single element representing the stack "εa
d adds the new nested stack as the innermost
ightmost) nested stack (<u>note:</u> This type of move
called a <u>create-move</u>).

v) $id_1 =(\alpha\cdot t_1\cdot t_2,\beta)$ and $id_2 = (\alpha\cdot k(t_1,t_2),\beta)$ wh
$\underline{shift}((t_1(\varepsilon),t_2(\varepsilon)),\cdot)$

is condition takes the innermost (rightmost) nest
ack t_2 and pushes (adds) the nested stack t_2 onto
p of the next innermost nested stack t_1 (<u>note:</u> T
pe of move will be called a <u>merge-move</u>).

) $id_1 = (\alpha\cdot t,\beta)$ and $id_2 = (\alpha\cdot k_2(k_1,t),\beta)$ where
$=\underline{shift}(\underline{start},\varepsilon)$ and $k_2=\underline{shift}((k_1,t(\varepsilon)),\cdot)$

other words, this condition takes the innermost
sted stack t and pushes (adds) the nested stack t
to a new empty nested stack creating a new nested
ack of one element where the element is the neste
ack t (<u>note:</u> This type of move will be called an
<u>bed-move</u>).

i) $id_1 = (t,\varepsilon)$ and $id_2 = (\underline{start},\varepsilon)$ where
$>s \in \underline{reduce}(t(\varepsilon))$ and $t \in skeleton(s)$

te that this condition is a reduce-move on the
p-level nested stack using the start (macro)
oduction $S\text{->}\theta$ where $lift(\theta)(\underline{|})=s$ defines the
rresponding start (tree) production $S\text{->}s$, and is
cause acceptance of the input string.

A $BUTLR_M(0)$ parser is considered deterministic
d only if for every instantaneous description id_1
ere exists an id_2 such that $id_1 \vdash_d^m id_2$, then id_2
ique. In other words, a $BUTLR_M(0)$ parser
$(G,TG_G,K,\underline{shift},\underline{reduce},\underline{goto},\underline{start})$ is <u>deterministi</u>
d only if

i) The BUTLR(0) parser $M' = (TG_G, K, \underline{shift}, \underline{reduce}, \underline{goto}, \underline{start})$ is deterministic.

ii) There are not shift/shift, shift/reduce, shift/create, shift/embed, shift/merge, reduce/reduce, reduce/create, ... , merge/merge conflicts.

Example **7.3.5:** Let G and TG_G be the macro gramma the corresponding tree grammar defined in exampl 7.3.3. Then $M=(G, TG_G, K, \underline{shift}, \underline{reduce}, \underline{goto}, \underline{start}$ $BUTLR_M(0)$ parser such that

$K = \{1,2,...,27\}$ where

$r(1) = r(2) = ... = r(7) = 0,$

$r(3) = r(4) = ... = r(16)$

$= r(18) = r(19) = ... = r(23)$

$= r(25) = r(26) = r(27) = 2,$

and $r(17) = r(24) = 3;$ and

shift, reduce, and goto are defined by the fo tables:

<u>shift</u>

	.	|	e	a	b	c	d
1		2	3	4	5	6	7
(3,4)	8						
(3,5)	9						
(3,6)	10						
(8,7)	11						
(9,7)	12						
(10,7)	13						
(8,11)	14						
(2,11)	15						
(9,12)	16						
(10,13)	18						
(14,7)	19						
(15,7)	20						
(16,7)	21						
(18,7)	22						
(8,19)	14						
(2,19)	15						
(20,12)	23						
(9,21)	16						
(20,21)	23						
(10,22)	18						

(continued on next page)

	.	⊥	e	a	b	c	d
(23,7)	25						
(25,13)	26						
(25,22)	26						
(26,7)	27						

<u>goto</u>

	F
(2,11,12,13)	17
(2,19,21,22)	24

reduce

```
    +--------------------------------+
    |   S ->        F                 |
    |                                 |
    |   ⊥      .      .      .       |
17  |         / \    / \    / \      |
    |        .   d  .   d  .   d      |
    |       / \    / \    / \         |
    |      ε   a  ε   a  ε   a       |
    +--------------------------------+
    |   F ->            F             |
    |                                 |
    |x y z w x      .      .      .  |
    |              / \    / \    / \ |
24  |             .   d  .   d  .   d|
    |            / \    / \    / \   |
    |           .   y  .   z  .   w  |
    |          / \    / \    / \     |
    |         ε   a  ε   b  ε   c    |
    +--------------------------------+
    |           F ->      .           |
    |                   / \          |
    |        x y z w   .   d          |
    |                 / \            |
    |                .   w            |
    |               / \              |
27  |              .   d             |
    |             / \                |
    |            .   z               |
    |           / \                  |
    |          .   d                 |
    |         / \                    |
    |        x   y                   |
    +--------------------------------+
```

The language accepted by the BUTLR$_M$(0) parser

set of strings

$$\{a^n d^{n+1} b^n d^{n+1} c^n d^{n+1} \mid n \geq 1\}$$

and by inspection of the tables above, clearly

deterministic. For example, the string

"aadddbbdddccddd" is accepted by the BUTLR$_M$(0)

follows:

```
                            m
, aadddbbdddccddd)  ⊢
                            d

                                m
• 8 , adddbbdddccddd)  ⊢
 / \                            d
3   4

                                        m
• 8   •  8 , dddbbdddccddd)  ⊢
 / \    / \                             d
3   4  3   4

                                        m
• 8   •   11 , ddbbdddccddd)  ⊢
 / \     /  \                           d
3   4  8     7
      / \
     3   4

                                  m
 •   14   , ddbbdddccddd)  ⊢
  /      \                        d
 8        11
/ \      /  \
3   4   8     7
       / \
      3   4

                                 m
 •   19   , dbbdddccddd)  ⊢
    / \                          d
  14   7
 /      \
8        11
 \      /  \
  4   8     7
     / \
    3   4
```

```
(    15       , dbbdddccddd)  ⊢^m_d
   /     \
 2        19
         /  \
       14    7
      /       \
    8          11
   / \        /  \
 3    4   8       7
         / \
        3   4


(         20    , bbdddccddd)  ⊢^m_d
         /  \
    15       7
   /     \
 2        19
         /  \
       14    7
      /       \
    8          11
   / \        /  \
 3    4   8       7
         / \
        3   4


(         20     ∘   9    , bdddccddd)  (⊢^m_d)^4
         /  \       / \
    15       7    3     5
   /     \
 2        19
         /  \
       14    7
      /       \
    8          11
   / \        /  \
 3    4   8       7
         / \
        3   4
```

```
                               2
  20     •     21   , dccddd) ⊢m
                              d
 /  \         /  \
5    7       16   7
  \        /    \
   19      9      12
  /  \   / \    /  \
 14   7 3   5  9    7
    \         / \
     11      3   5
    /  \
4  8    7
  / \
 3   4
```

```
                              7
            25      , ccddd) ⊢m
                             d
          /    \
       23        7
    /      \
 20          21
/  \        /  \
    7      16    7
 \        /   \
  19     9      12
 /  \   / \    /  \
14   7 3   5  9    7
   \         / \
    11      3   5
   /  \
  8    7
 / \
3   4
```

```
(                           27   , ε)  ⊢_d^m
                          /    \
                       26       7
                    /      \
                 25           22
               /    \        /  \
            23       7      18   7
         /      \         /    \
      20          21     10     13
     /  \        /  \   /  \   /  \
   15    7      16   7 3    6 10   7
 /    \        /   \         /  \
2      19     9      12     3    6
      /  \   / \    /  \
     14   7 3   5  9    7
   /    \         / \
  8      11      3   5
 / \    /  \
3   4  8    7
      / \
     3   4
```

```
(               24                   , ε) ⊢_d^m
   /       /       \        \
2       19           21           22
      /  \         /  \         /  \
     14   7       16   7        18  7
   /   \        /   \         /   \
  8      11    9      12     10     13
 / \   /  \   / \   /  \   /  \   /  \
3   4 8    7 3   5 9    7 3    6  10  7
     / \          / \           /  \
    3   4        3   5         3    6
```

```
(        F         ,  ε) ⊢_d^m
  /    /   \    \
2    11     12     13
   /  \   /  \   /  \
  8    7 9    7  10  7
 / \    / \    /  \
3   4  3   5  3    6
```

(start,ε)

Like the $BUTLR_S(0)$ parser, there are several
njectures the author has about this model and are
llows:

<u>njecture</u> <u>7.3.1:</u> Given a macro grammar G and the
$TLR_M(0)$ parser $M=(G,TG_G,K,$<u>shift</u>,<u>reduce</u>,<u>goto</u>,<u>start</u>
$_I(G)=N(M)$.

<u>njecture</u> <u>7.3.2:</u> Given a macro grammar G, the
$TLR_M(0)$ parser can be extended to a $BUTLR_M(k)$ par
parser with k symbols of lookahead on the input
pe).

<u>njecture</u> <u>7.3.3:</u> Given a string grammar G, the LR(
rser $M_1=(G,K_1,$<u>shift</u>$_1$,<u>reduce</u>$_1$,<u>goto</u>$_1$,<u>start</u>$_1$), and t
$TLR_M(0)$ parser
$=(G,TG_G,K_2,$<u>shift</u>$_2$,<u>reduce</u>$_2$,<u>goto</u>$_2$,<u>start</u>$_2$), then

a) (<u>start</u>, α) $\vdash_d^n$ (β,θ) if and only if
(<u>shift</u>$_2$(<u>start</u>,$\underline{\perp}$), α) $\vdash_d^{m^n}$ (t,θ) where
<u>spelling</u>(β)=yield(<u>spelling</u>(t)) and
lift(<u>spelling</u>(β))($\underline{\perp}$)=<u>spelling</u>(t).

b) M_1 is deterministic if and only if M_2
deterministic

c) $L(G)=N(M_1)=N(M_2)$

Conjecture 7.3.4: Given a string grammar G, th $BUTLR_M(0)$ parser can be extended to a $BUTLR_M$(k (a parser with k symbols of lookahead on the i tape). Furthermore, the conditions of conject apply to the comparison between the LR(k) pars the $BUTLR_M(k)$ parser.

Conjecture 7.3.5: The class of string language recognized by the class of deterministic LR(k) languages is a subclass of the class of langua recognized by the class of deterministic BUTLR parsers. Furthermore, the inclusion is proper

While this chapter has presented a new pa model for the class of macro languages and the deterministic model is quite powerful, it may the most powerful form of a parsing model for of macro languages. One may have wondered why language $\{a^n d^{n+1} b^n d^{n+1} c^n d^{n+1} \mid n \geq 1\}$ was chosen example instead of the string language $\{a^n b^n c^n$

reason for this is that the second case will uce a nondeterministic $BUTLR_M(0)$ parser and the e is that the $BUTLR_M(0)$ parser does not take ntage of the context of the tree stacks occurring he left of the tree stack being updated. In othe s, whenever a new nested stack is created, the ext of all other outer nested stacks is lost. e, a possible way to increase determinism is to fy the BUTLR(0) construction method to use left ext in the same manner that the LR(0) parsing od does in simulating a bottom-up tree automaton e the stack is a list of current states associat read-heads occurring to the left of the read-he g updated, and the update is based on the conten he stack.

Chapter VIII

CONCLUSION

Chapter two presented the notation and ter used in this thesis. Chapter three presented context-free (string) languages and a summary o LR(0) parsing techniques. Chapter four present context-free tree languages and several results context-free tree languages which are based on known results about context-free (string) gramm Chapter five presented the tree pushdown automa which accepts the class of context-free tree la and chapter six presented a construction method build a deterministic tree pushdown automaton (BUTLR(0) parser) for a subclass of the context-

ree languages. Finally, chapter seven presented
pplication of the BUTLR(0) parser to parse strin
anguages in the class of OI macro languages. Th
hapter provides a summary of the major results o
issertation in terms of its contribution to comp
cience, as well as open questions and possible f
esearch on the topics covered in this thesis.

.1 Summary Of Research

The crux of this dissertation is to present
arsing model to recognize the class of context-f
ree languages using a new parsing model which is
ottom-up tree automaton augmented with internal
onsisting of a finite sequence of trees (called
tacks). This new form of automaton is called a
ushdown automaton. Furthermore, the tree pushdo
utomaton corresponds to the standard (string) pu
utomaton in the same manner that the bottom-up t
utomaton corresponds to the (string) finite auto
ence, like the pushdown automaton, the tree push
utomaton can only access the tree stack through
oot, and nodes can only be pushed (added) or pop
(deleted) at the root of a tree stack.

The goal of this dissertation is to develo types of parser constructors. The first type o constructor is a constructor to build a determi tree pushdown automaton which has the power to recognize a subclass of the context-free tree languages, and is based on the notions used by parser. The second type of parser constructor parser constructor which takes the first parser constructor and applies its construction method strings in order to obtain a new parser constru which would have the power to recognize the mac (string) languages (which is more general than class of context-free string languages on which LR(0) parser is based on). One should note tha first type of parser constructor is the major g this thesis while the second is to provide an application of the first.

The ideas and inspiration used throughout thesis was to mimic and generalize the construc methods used by LR(k) parser constructors, and so, develop a new, more powerful parsing techni this end, the methods of the LR(k) parser const are lifted to a more powerful form of languages as context-free tree languages, and the develop a construction method which creates a determini arser for a subclass of the context-free tree anguages.

The major results of this dissertation in ter t meeting the above goals are threefold: (1) The lass of tree languages generated by context-free anguages is identical with the class of tree lang ecognized by tree pushdown automata; (2) The the f LR-parsing (shift-reduce parser constructors fo ontext-free string languages) extends to context- ree languages. More specifically, the natural eneralization of the LR(0) parser constructor is UTLR(0) parser constructor which generates a clas arsers that recognizes the class of context-free anguages and builds a deterministic parser for a ubclass of the context-free tree languages; and he author conjectures that the BUTLR(0) parser ca sed to build a parser to recognize macro (string) anguages (the $\mathrm{BUTLR}_M(0)$ parser) and the construct ethod constructs a deterministic parser for a sub f the macro languages. Furthermore, the class of tring languages recognized by the deterministic $\mathrm{UTLR}_M(k)$ parser should be a proper superclass of lass of string languages recognized by determinis R(k) parsers.

The key to both the BUTLR(0) parser and th
$BUTLR_M(0)$ parser is the characteristic automato
the LR(0) parser, the BUTLR(0) parser directly
from its characteristic automaton. The charact
automaton of the BUTLR(0) parser is an automato
to recognize deterministically a set of charact
trees, which are well defined subtrees of the
sentential forms generated by a tree grammar.
Furthermore, the construction of the characteri
automaton is based on the fact that the set of
characteristic trees can be generated by a root
tree grammar and hence, the set of characterist
correspond to a co-regular tree language.

To summarize the major results of this
dissertation in terms of the construction of th
BUTLR(0) parser, and the tree pushdown automato
the BUTLR(0) parser is based on, the following
results is presented:

1) The power of the tree pushdown automat
the same as the power of a context-fre
grammar (see theorem 5.9.1). In other
the class of tree languages accepted b
pushdown automata is identical to the
(OI) context-free tree languages.

In general, the set of characteristic trees generated by context-free tree grammar is not a regular tree language (see theorem 6.2.1).

Given a context-free tree grammar G, a root-linear tree grammar G′ can be constructed such that the tree language generated by G′ is identical to the set of characteristic trees of G (see theorem 6.2.2). As a consequence, the set of characteristic trees generated by a context-free tree grammar is a co-regular tree language.

The class of co-regular tree languages is a proper subset of the class of context-free tree languages (see theorem 4.11.3). Hence, recognizing characteristic trees should be simpler than recognizing context-free tree languages.

The construction method of the characteristic automaton for a context-free tree grammar G produces a bottom-up tree automaton M such that M recognizes every characteristic tree generated by G. However, it is not necessarily the case that the set of trees recognized by the characteristic automaton M

is identical to the set of characteris
trees (see theorems 6.2.3 and 6.2.4).

6) A BUTLR(0) parser M, constructed from context-free tree grammar G, recognize exactly the tree language generated by Furthermore, for a subclass of the context-free tree languages, the const BUTLR(0) parser is deterministic (see 6.3.1).

Furthermore, while producing the above res about tree pushdown automata and the BUTLR(0) construction method, the following related resu also been shown:

1) Given a tree grammar G, the tree langu generated under an OI derivation mode identical to the tree language generat a prefix lexicographical ordering deri mode (i.e. a leftmost OI derivation, theorem 4.1.2).

2) Like context-free (string) grammars, given a context-free tree grammar G, one can effectively construct a context-free tree grammar G′ such that the tree languages generated by G and G′ are identical, and G′ i in 2-normal form (see theorem 4.5.1). In other words, the right-hand side of every production in G′ contains at most a total of nonterminal or terminal symbols.

3) Like context-free (string) grammars, given a context-free tree grammar G, one can effectively construct a context-free tree grammar G′ such that the tree languages generated by G and G′ are identical, and G′ i in "weak Chomsky normal form" (see theorem 4.9.1). In other words, the tree grammar G′ is in 2-normal form where the right-hand side of every production in G′ contains either

i) 2 nonterminal symbols and variables

ii) a single terminal symbol and variables

iii) a one node tree labeled by a vari

4) A "pumping lemma" for the class of co tree languages (see theorem 4.11.2). lemma can be uses to show that a part tree language is not in the class of co-regular tree languages.

8.2 Open Questions

While working on the above topics, severa topics and concepts have been brought up for w solution has been found. These topics represe questions in this field of study (and related and can be broken down into two types of quest This first type of open questions are those fo the author has conjectured answers to the ques but has not found a proof of the result, and a presented by the following list:

1) The class of tree grammars for which deterministic BUTLR(0) parser will be constructed is the class of BUOI(0) t grammars (where BUOI(0) corresponds t meaning of LR(0) grammars for context string languages, see section 6.4).

Given a macro grammar G, the string language generated under an OI derivation mode is identical to the string language generated under a <u>rightmost</u> OI derivation mode (see conjecture 7.2.1).

) Given a macro grammar G and the $\text{BUTLR}_M(0)$ parser M generated by G, the string language recognized by the $\text{BUTLR}_M(0)$ parser M is identical to the string language generated by the macro grammar G under an OI derivation mode (see conjecture 7.3.1).

) The $\text{BUTLR}_M(0)$ parser constructor can be extended to a $\text{BUTLR}_M(k)$ parser constructor (a parser with k symbols of lookahead on the input tape, see conjecture 7.3.2).

) Given a context-free (string) grammar G, the $\text{BUTLR}_M(0)$ parser M generated by G, and the LR(0) parser D generated by G:

i) The set of possible moves that can be made by the $\text{BUTLR}_M(0)$ parser M is identical to the set of possible moves that can be made by the LR(0) parser D (i.e. the $\text{BUTLR}_M(0)$ parser M simulates the LR(0) parser D).

ii) THe $BUTLR_M(0)$ parser M is determin

and only if the LR(0) parser D is

deterministic.

iii) The string language generated by t

string grammar G is identical to t

string language recognized by the

$BUTLR_M(0)$ parser M, which is ident

the string language recognized by

LR(0) parser D.

(see conjecture 7.3.3)

6) The conditions of 5 (above) also apply

comparisons between the $BUTLR_M(k)$ pars

the LR(k) parser.

7) The class of string languages recogniz

the class of deterministic LR(k) parse

subclass of the class of string langua

recognized by the class of determinist

$BUTLR_M(k)$ parsers. Furthermore, the i

is proper.

The second type of open question are those that

brought up by the author, but for which the aut

cannot even make a conjecture. to the question

1) Can epsilon rules (productions with just a single variable on the right-hand side) be removed from context-free tree languages (s section 4.7)?

2) Is there an effective method of converting nonconservative tree grammars (i.e. tree grammars which contain a production where a variable occurs on the left-hand side of th production but not the right) into conservative tree grammars (see section 4.7

3) Can a tree grammar be reduced? That is, gi any tree grammar G, can the tree grammar G converted to a tree grammar G′ such that th tree language generated by G and G′ are identical, and every production in G′ is us in the generation of some tree in the tree language generated by G (see section 4.8).

4) Does there exist a pumping lemma for the cl of context-free tree languages?

5) Does there exist a parsing model which recognizes exactly the class of co-regular tree languages? Furthermore, does there ex a parser constructor for this model such th

the construction will guarantee to pro[...]
deterministic parser for the class of
co-regular tree languages (see section

6) Can the definition of the tree pushdow[...] automaton be converted to use unificat[...] instead of "simple" tree matching and [...] remove the restriction that the tree p[...] automaton will not (by default) be nondeterministic whenever the tree gra[...] which generated the tree pushdown auto[...] nonconservative.

8.3 Future Research

Future research in this area could follow [...] different directions. At one end of the spectr[...] research could continue along the theoretical v[...] investigating other parsing models to recognize languages as well as possible generalizations ([...] restrictions) on the type of tree grammar used. to the other end of the spectrum, research coul[...] continue by investigating the application side, how the tree pushdown automaton (and the BUTLR([...] parser constructor) can be applied to macro (st[...] languages. The following paragraphs state some

:hor's continuing interests in this area in both
eory and application.

One research goal is to investigate
ieralizations of the BUTLR(0) parser constructor
ler to increase the class of tree languages
:ognized deterministically. For example, the not
a lookahead should be introduced to produce a
[LR(k) parser which should reduce nondeterminism.
e should note that this includes two different ty
lookahead where the meaning of K symbols of
okahead is dependent on whether or not the input
iguage will be a string or tree language. In the
ring case, the k symbols of lookahead corresponds
e next k leaves (or symbols on the input) while i
e tree case, the k symbols of lookahead correspon
the next k immediate ancestors of the node. Lik
rsing techniques, other possible modifications to
vestigated are construction methods which lift th
tions of SLR(1) and LALR(1) parser constructors (
Remmer[69,71,72], Anderson, Eve, and Horning[72],
Londe, Lee, and Horning[71]) to trees in order to
oduce "smaller" deterministic parsers for a subcl
the tree languages recognized by the class of
terministic BUTLR(k) parsers. Still other possib
difications to the BUTLR(k) parser constructor to

investigated are state minimization methods lik of compatibility for LR(k) parsers introduced b Pager[77a,77b] which will produce an equivalent to that of a BUTLR(k) parser, but requires less and hence, less machinery.

Another mode of research will be to invest other types of tree representations (besides th graphical representation of Gorn trees used in thesis) and the corresponding effects on the ma needed. For example, one possibility is to lin tree structures by using some type of total ord the nodes of the input tree (postorder for exam Intuitively, such an ordering should simplify t model from a tree pushdown automaton to a neste automaton (a finite-state automaton augmented w internal memory which is a self-stacking pushdo stack, see Aho[69]).

Yet another goal of future research is to investigate some of the other eight possible fo tree pushdown automata (as mentioned in the introduction). In particular, there is another of the tree pushdown automaton envisioned by th which will accept the class of IO context-free languages (which is a different class of tree l han the class of OI context-free tree languages, ngleFriedt and Schmidt[77,78]). The model consis top-down tree automaton that uses a <u>single</u> tree o remember the structure of the scanned portion o nput tree. Like a BUTLR(0) parser, there is also nvisioned parser constructor which also resembles R(k) parser constructors and should construct a eterministic model for a large subclass of the IO ontext-free tree languages.

As with the BUTLR(0) parser, the envisioned onstructor starts by building a deterministic top utomaton to recognize characteristic trees. From utomaton, the <u>shift</u>, <u>reduce</u>, and <u>goto</u> tables dire reated and define the generated parser. However, nlike the BUTLR(0) parser, included in the defini f shift-moves and reduce-moves is the existence o parallel" action where the action is applied to <u>a</u> ubtrees of the input tree which must represent th ame tree (or value as an IO derivation mode impli ts parameters).

Another possibility is to study some relaxati n the definition of context-free tree languages, he effects these alterations have on the parsing odels. In particular, the type of relaxation to investigated is the removal of the restriction t every production $F(x_1,...,x_n)$->t, only variables through x_n can occur in the tree t. That is, to introduce the notion of "local" variables is to introduced into productions which will allow the specification of a tree language using a tree gr which only partially describes the structure of trees in the language (unlike context-free tree grammars which totally specify the tree language Questions of interest are both in the interpreta these "local" variables, and how the eight poss forms of tree pushdown automata can be modified handle such types of language specification.

Another goal is to investigate the $BUTLR_M$(0 parser in more detail than was done in this thes This includes investigating generalizations or modifications to both the BUTLR(0) and $BUTLR_M$(0) constructors in order to increase the class of s languages that will be recognized by determinist $BUTLR_M$(0) parsers. In particular, to investigat to add left-to-right context such that <u>every</u> mov by the $BUTLR_M$(0) parser is based on <u>all</u> the info gained so far (which is currently lost whenever nested stack is created).

second area of application, besides parsing the (string) languages, is along the lines of ng an automatic compiler constructor. That is, ress both the syntax and semantics of the er using two types of rewrite systems, and from rewrite systems construct automatically the er. The first rewrite system would be a string r describing the syntax which would be used to the input and produce a parse tree as is tly done with LR parsers. The second rewrite would be a tree grammar defining a tree ucer where the tree transducer would be used to t the parse tree of the input into the ically equivalent tree in the object language.

n conclusion, the area appears to be very rich in ch possibilities for the forseable future. The believes that the parsing models introduced in hesis will result in practical models, sometime future, which will be used by the computer e community.

INDEX

BIBLIOGRAPHY

A. V. [1969] - Nested Stack Automata. JACM, ol. 16, no. 3, July 1969, pp. 383-406

A. V. and Ullman, J. D. [1972] - The Theory arsing, Translation, and Compiling. Internation . Computer Mathematics 3, p.149-155

A. V. and Ullman, J. D. [1979] - Principles ompiler design. Addison-Wesley Publishing C eading Mass.

rb, Bowen, Martin Chaney, Micheal Fay, homas Pennello, and Rachel Radin [1976] ranslator writing system for the Burroughs B570 nformation Sciences, UC Santa Cruz, Santa Cruz, C

rson, T.; Eve, J.; and Horning, J. [1973] - fficient LR(1) parsers, Acta Informatica 2, 2-39

, M. A., Kfoury, A. J., and Moll, R. N. [198 basis for theoretical computer science, Spring erlag, New York, 1981

ld, A. and Dauchet, M. [76] - Un theoreme de uplication pour les forets algebriques, JCSS, v 3, #3 (October 1976), p.223-244

Hillel Y.; Perles, M.; and Shamir, E. [1961] - n formal properties of simple phrase-structu rammars, Zeitschrift fur Phoneti prachwissenschaft, und Kommunikationsforshung, 1 .143-177

Hillel, Y. and Shamir, E. [1960] - Finite-state anguages: formal representations and adequa roblems, The Bulletin of the Research Council sreal, 8F, p.155-166

y, G.; Levy, J. [1978] - Minimal and Optimal omputation of recursive programs. JACM vol 26, , p. 148-175, January 1978

Brainerd, W. S. [1969] - Tree generating regu
systems, Informationand Control 14, 217-231

Brosgol, B. M. [1974] - Deterministic transla
grammars, PhD Thesis, Technical report #
Center for Research In Computing Technology,
University, Cambridge Mass

Buchi, J. R. [1960] - Weak second order arith
and finite automata, Zeit. Math. Log
Grundlagen. Math. 6, p. 66-92

Buchi, J. R. and Elgot, C. C. [1958] - Deci
problems of weak second-order arithmetics an
automata. Abstract 553-112, Notices Amer
Soc. 5, 834

Buchi, J. R. and Wright, J. B. [1960] -
Mathematical theory of automata. Notes on
presented by J. R. Buchi and J. B.
Communication Sciences 403, Fall 1960, Unive
Michigan, Ann Arbor, Michagan

Chomsky, N. [1956] - Three models for the
descriptions of Languages, IRE Transacti
Information Theory 2, #3, p.113-124

Chomsky, N. [1959] - On certain formal propert
grammars, Information and Control, 2, p.137-

Chomsky, N. [1962] -Context-free grammars and
pushdown storage, MIT Research Lab of Ele
Quarterly Progress Report 65

Chomsky, N. and Miller, G. A. [1958] - Finit
languages, Information and control, 1, p.91-

Courcelle, B. [1976] - Completeness results fo
equivalence of recursive schemes. ^J.
System Sci._12(2), p.179-197

Courcelle, B. [1981] - A representation of tre
languages, I and II, ^theor. Comput. Sci.\

eikis, F. and Ripley, G. D. [77] - Extended LR(k) parsers for error recovery and repair. Dep f Computer Science, Univ. of Arizona, Tuscan Az.

mer, F. [69] - Practical translators for LR(k) anguages, Ph.D. thesis, Dept. of Electric ngineering, M.I.T., Cambridge, Mass.

mer, F. L. [71] - Simple LR(k) grammars. Comm. CM 14, p. 453-460

mer, F. [82] - XPL distribution tape containing ALR translator writing system. Computer a nformation Sciences, UC Santa Cruz, CA.

r, J. E. [70] - Tree acceptors and some of thei pplications, J. Comput. System Sci. (JCSS) 4

ey, P. J. [74] - Formal Languages and ecursion Schemes, PhD dissertation, Harva niversity, Cambridge Massachusetts.

e, J., Parchmann, R., and Specht, J. [79] zilard languages of IO - grammars, Information a ontrol 40, no.3, pp. 319-331

nberg, S. [74] - Automata, Languages, and achines, Vol. A, Academic Press, New York

nberg, S. and Wright, J. B. [67] - Automata ir eneral algebras, Information and Control 11: .452-470

t, C. C. [61] - Decision problems of finite utomaton design and related arithmetics. Tran mer. Math. Soc. 98, p.21-51

lfriet, J. [80] - Some open questions and ecent results on tree transducers and tı anguages, Memorandum nr. 293, Technisc ogeschool twente, Enschede, Netherlands

lfriet, J. [80] - Some open questions and ecent results on tree transducers and tı anguages, in Formal Language Theory (R. Bo ditor), Academic Press

Engelfriet, J. [81] - Tree transducers and syntax-directed semantics, Memorandum nr Technische Hogeschool twente, Enschede, Net

Englefiet, J. [75] - Bottom-up and Top-down t transformations - a comparison, Technical U twente, Enschede, Netherlands

Engelfriet, J.; Schmidt, E. M. [77] - IO an ^J. of computer and system sciences_15, p

Engelfriet, J.; Schmidt, E. M. [78] - IO an ^J. of computer and system sciences_16, p

Evey, R. J. [63] - The theory and applicatio pushdown store machines, PhD. Thesis and Report, Mathematical Linguistics and Translation Project, Harvard University, NS

Fischer, M. J. [68] - Grammars with macro-li productions. Doctoral Dissertation, University, Cambridge Mass.

Fischer, M. J. [1969] - Grammars with macro- productions. 9th Symposium on switch automata theory.

Friedman, E. P. [76] - The inclusion problem simple languages, ^Theor. Comput. S p.279-316

Friedman, E. P. [77a] - Equivalence Problems Deterministic Context-Free Languages and Recursion Schemes, ^J. Comput. System Sci 344-359

Friedman, E. P. [77b] - Simple Context-Free Languages and Free Monadic Recursion Scheme Systems Theory_11, p. 9-28

Gallier, J. H. [80] - Deterministic Finite A with Recursive Calls and DPDA's (detailed a technical report #MS-CIS-80-36, Dept. University of Pennsylvania lier, J. H. - DPDA'S in "Atomic Normal Form" and applications to equivalence proble Theoretical Computer Science 14, North-Holl Publishing Co., 155-186

ler, M. M. and Harrison, M. A. [77] - Characteristic parsing: A framework for produc compact deterministic parsers I., JCSS 14, p.265-

sburg, S [66] - The mathematical theory of context-free languages, McGraw-Hill Book Co., York

sburg, S. and Greibach, S. A. [66] - Deterministic context-free languages, Informat and Control 9, p.620-649

n, S. [1962] - Processors for infinite codes of Shannon - Fano type, ^Proceedings of the Sympos on Mathematical theory of Automata\

n, S. [1962] - Symbolic Languages in d cessing
Proceedings of the Symposium organized and editec the International Computation Centre Rome, Ma 26-31, Gordon and Breach Science Publishers, N.Y.

n, S. [1965] - Explicit Definitions and Linguist Dominoes, ^Systems and Computer Science\, John F and Satoru Takasu Eds.

n, S. [1967] - Handling the growth by definition mechanical languages, Spring Joint Compu Conference, 1967

es, D. [71] - Compiler construction for digital computers, John Wiley and Sons, New York (1971)

ssarian, I. [81] - On pushdown tree automata. Proceedings of 6th CAAP (Genoa, lecture notes computer science, Springer-Verlag) 1981

tag, J., Horowitz, E., and Musser, D. [76] - Abstract data types and software validation, report 76-48, University of California, Los Angel

Guttag, J., Horowitz, E., and Musser, D. [78] - Abstract data types and software validation 21, #12, p.1048-1064

Haines, L. H. [65] - Generation of Recognition Formal Languages, PhD Thesis, MIT (1965)

Harrison, M. A. [78] - Introduction to formal language theory, Addison-Wesley Publishin Reading Mass. 1978

Harrison, M. A. and Havel, I. M. [73] - On a of deterministic grammars, in Automata, Lan and Programming (M. Nivat ed.), p.4 North-Holland Publishing Co., Amsterdam (1973

Hoffman, M. and O'Donnell, M. J. [82] - Patte matching in trees, JACM 29, #1, p.68-95

Hopcroft, J. E. and Ullman, J. D. [69] - For languages and their relation to au Addison-Wesley Pusblishing Co. (1969)

Huet, G. [80] - Confluent Reductions: Abstract propoerties and applications to term re systems. JACM, vol 27, no 4, october 1980

Huet, G.; Oppen, D. C. [80] - Equations and r rules, A survey. "Formal Languages: Persp and open problems". Editor, Ron Book, A Press, 1980

Joshi, A. K.; Levy, L. S. [77] - Constraints structural descriptions : local transform SIAM ^J. of Computation\, vol 6, no2, June 1

Joshi, A. K.; Levy, L. S.; Takahashi M. [75 Tree Adjunct Grammars, ^J. of Computer and Sciences\, Vol 10,#1 Febuary 1975

Knuth, D. E. [65] - On the translation of lang from left to right. Information and Control 607-638 th, D. E. [68] - Semantics of Context-free Languages. ^Mathematical Systems Theory\, vol#2, 2 (1968)

th, D. E. [71] - Top-down Syntac Analysis, Acta Informatica 1, #2, p.79-110

stensen, B. B.; Madsen, O. L. [81] - Methods Computing LALR(k) lookahead. ACM transactions programming languages and systems, Vol 3, No January 1981

onde, W. R.; Lee, E. S.; and Horning, J.]
- An LALR(k) parser generator. Proc. I congress71, North Holland, Amsterdam, p.#151-153

onde, W. R. and Rivieres, J. D. [81] - Handli operator precedence in arithmetic expressions w tree transformations. ACM Transactions Programming Languages and Systems, Vol 3, January 1981, p.83-103

tin, A. and Schutzenberger, M. P. [67] - A combinatorial problem in the theory of free monoi Combinatorial Mathematics and its Applications, 128-144 (1967)

is, H. R.; Papadimitriou, C. H. [81] - Elemen of the theory of computation, Prentice Ha Englewood Cliffs, N.J. 1981

is II, P. M. and Stearns, R. E. [68] - Syntax-directed Transduction, JACM 15, p.465-488

idor, M. and Moran, G. [69] - Finite automata over finite trees, Technical Report 30, Heb University, Isreal

lhorn, K. [79] - Parsing Macro Grammars top down, Information and Controo 40, no.1, 123-143 lner, R. A. [78] - A theory of type polymorphis in programming, JCSS 17, #3 (December 19 p.345-375

vat, M. [75] - On the interpretation of recursiv polyadic program schemes, Symposia Mathematica Academic Press, N.Y., 225-281 (1975)

ttinger, A. G. [61] - Automatic Syntactic Analy and the Pushdown Store, Proceedings of Symposia Applied Mathematics (vol 12), American Mathemat Society, Providence, R.I. (1961)

ger, D. [77a] - A practical general method for constructing LR(k) parsers. Acta Informatica 7, 249-268 (1977)

ger, D. [77b] - The lane tracing algorithm for constructing LR(k) parsers and was of enhancing efficeincy. Information Sciences 12, p.#1 (1977)

ather,R. E. [76] - Descrete mathematical struct for computer science, Hougton Mifflin Co., Bo 1976

rdom, P.; Brown, C. A. [79] - Parsing extended LR(k) Grammars, Technical report #87, comp science dept., University of Indiana. 1979

rdom, P.; Brown, C. A. [80] - Semantic Routine and LR(k) Parsers, Acta Informatica, (1980)

bin, M. O. and Scott, D. [59] - Finite automat and their decision problems. IBM Journal Research and Development, 3, p.114-125 (1959)

bin, M. O. [68] - Decidability of second order theories and automata on infinite trees. Research Rept. RC#2012. IBM Yorktown Heights, (1968)

sen, B. K. [73] - Tree-manipulating systems and Church-Rosser Theorems. JACM 20, 160-188 (1973)

nds, W. C. [69] - Context-free grammars on tree
IEEE annual Symp. Switching and Automata Theo
10th, Oct. 1969, pp 143-148

nds, W. C. [70] - Mappings and grammars on tree
J. Math System Theory 4, #3, pp. 257-287

omaa, A. [69] - Theory of Automata,
Pergamon Press, New York (1969)

omaa, A. [73] - Formal Languages, Academic Press
New York (1973)

impf, K. M. [81] - Construction Methods of LR
Parsers, master's thesis, technical rep
#MS-CIS-80-40, Dept. of CIS, University
Pennsylvania, 1981

utzenberger, M. P. [63] - On context-free
languages and pushdown automata, Information
Control 6, p.246-264

tcher, J. W. [67] - Characterizing derivation
trees of context-free grammars through
generalization of finite automata theory. JCSS
316-332 (1967)

tcher, J. W. [73] - Tree automata: An informal
survey, in Currents in the Theory of Computing
V. Aho ed.) Prentice-Hall series in automa
computation, Prentice-Hall Inc., Englewood Clif
N.J. P.143-172 (1973)

tcher, J. W. and Wright, J. B. [68]
- Generalized finite automata theory with
application to a decision problem of second or
logic, J. Math. Systems Theory 2 (1968)

d M. [77] - Algebraic theories and tree rewritin
systems, Technical rept. 66, Dept. of Compu
Science, Indiana University, Bloomington Ind.