U-TREE AUTOMATA:
MACHINES THAT CAN
CLASSIFY PATTERNS

Thesis for the Degree of Ph. D.
MICHIGAN STATE UNIVERSITY
KENNETH LEROY WILLIAMS
1973

ABSTRACT

U-TREE AUTOMATA:
MACHINES THAT CAN CLASSIFY PATTERNS

By

Kenneth Leroy Williams

A syntactic method for pattern recognition using an extension of standard tree automata theory to a theory for unordered trees is developed. A formalization is given for regular unordered tree grammars and automata. It is shown that regular unordered tree automata can serve as pattern classification devices. Many concepts inherent to syntactic pattern recognition methods are defined. The important differences between circular and noncircular primitive systems are noted. An exploration is made concerning what types of pattern classes used with what types of primitive systems are amenable to this method of classification. It is shown that deterministic pushdown automata can be used to simulate regular tree automata which in turn can be used to simulate regular unordered tree automata. The interesting class of languages generated as the frontier by regular unordered tree grammars is investigated. A characterization for various types of tree generating grammars based on the substitutions made in root-to-frontier paths is proposed.

U-TREE AUTOMATA:
MACHINES THAT CAN CLASSIFY PATTERNS

By

Kenneth Leroy Williams

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1973

ACKNOWLEDGEMENTS

I want to thank each of the members of my doctoral committee for his unique contributions. Thanks go to Dr. Herbert Bohnert for crossing department and college lines and serving on the committee as a member of the Philosophy Department (but a member with a great deal of computing background). Thanks go to Dr. Hans Lee for serving as my academic advisor and for continually trying to get me to see the "big picture" and integrate my somewhat narrow field of specialization into a broader context. Thanks go to Dr. Harry Hedges and the Department of Computer Science, as well as the Division of Engineering Research, for providing me with financial assistance during my doctoral studies. Thanks also are due to Dr. Hedges for his fine reading of earlier thesis drafts and pointing out many previously overlooked errors.

Of course the committee member and faculty member I am most indebted to, is my thesis advisor, Dr. Carl Page. Most of the computer theory that I know was learned from him; either in the many classes I enjoyed where he was the instructor or as a result of his encouragement to carry my research forward.

Special thanks are due to my wife Bonnie. Without her love and support I would never have reentered the academic world to pursue my doctorate.

TABLE OF CONTENTS

GLOSSARY

Term	Meaning	See Page
α/a	Subtree of α at a	10
A^*	All strings with symbols from set A	
A^+	All non-empty strings with symbols from set A	
A^n	All strings of length n with symbols from set A	
$A_1 \times A_2$	Cartesian product of sets A_1 and A_2	
$A \subseteq B$	A is a subset (or subclass) of B	
$A \subset B$	A is a proper subset (or subclass) of B	
$A \not\subseteq B$	A is not a subset (or subclass) of B	
$a \in A$	a is an element of set A	
$a \notin A$	a is not an element of set A	
$a \Rightarrow b$	b is derived from a in one step (or a implies b)	15,31
$a \overset{*}{\Rightarrow} b$	b is derived from a	15,31
$\overline{A}$	The complement of set A	
$a \leq b$	(Defined on page 9)	9
A_n	$\sigma^{-1}(n)$	9
$\langle A, \sigma \rangle$	Ranked alphabet A	9
a:x	A node of a d-tree	23
$\alpha(a \leftarrow \beta)$	(Defined on page 14)	14
b/a	(Defined on page 10)	10
$c_i, c_i^{-1}, c_i', c_i'^{-1}$	All mappings for $1 \leq i \leq 5$	28
Δ	Context-free sets (in Chapter 5 only)	86
d(a)	Depth of node(a)	9
$(D, \equiv)$	An equivalence relation	
f^{-1}	Inverse of mapping f	

Glossary

Term	Meaning	See Page
$\overline{f}$	All permutations on string f	31
Γ	String languages of regular u-tree grammars (in Chapter 5 only)	86
$\sim$	goes to or contains	12
LHS	Left hand side	
λ	String of length 0	
L(G)	Language of G	
N	Non-negative integers	
N^+	Positive integers	
0^n	The string composed of n (0)s	
$\Pi d(G)$	Projected derivation trees	58
$\rho(\alpha)$	Response to α	12,32
RHS	Right hand side	
RUTG	Regular u-tree grammar	30
RTG	Regular tree grammar	15
$\ni$	Such that	
$\|x\|$	Number of symbols in x	
$\leftarrow$	Replaced by	14
Θ	Regular sets (in Chapter 5 only)	86
#	Pushdown stack start symbol	74
2^Q	All subsets of set Q	
TA	Tree automaton	12
T_Σ	All trees over alphabet Σ	9
τ_Σ	All pseudoterms over Σ	11
$\phi\rightarrow\psi$	Tree (or u-tree) production	15,30

Glossary

Term	Meaning	See Page
t_x	Means t_i for $x=x_i$	12
U	Universal tree domain	8
UTA	U-tree automaton	32
$\langle U,V \rangle$	Digraph with U = set of nodes, V = set of edges	23
WOLOG	Without loss of generality	

CHAPTER 1

INTRODUCTION

1.1 MOTIVATIONS

Most pattern recognition schemes fall into one of two classes. The more common type involves feature extraction followed by statistical classification. Within the last several years, however, more and more attention has been devoted towards applying techniques of formal language theory to recognizing patterns. Such approaches are usually called syntactic or linguistic pattern recognition. A good introduction to the syntactic approach is given in Fu and Swain(18). Syntactic methods are appealing since they provide a vehicle for formalizing both the representation of patterns and the corresponding classification methods. A major disadvantage of the syntactic approach is that language theory has customarily dealt with one-dimensional strings of symbols and sets of such strings as its primary objects of study. Meanwhile most pattern recognition has been concerned with patterns as represented by two-dimensional pictures or other high-dimensional representations. Strings that are amenable to syntactic techniques are usually inadequate representations for high-dimensional patterns.

Interest in a wide variety of types of machines that may be called multidimensional automata also has developed within the last ten years. The multidimensional automata that have been best characterized and developed are those known as tree automata. Tree automata theory includes the study of tree grammars, tree transformation systems, tree languages and tree acceptors or pseudoautomata. It seems natural to try to apply some of the results of tree automata theory towards pattern recognition. The first attempt in this direction was that given by Fu and Bhargava(20). Fu and Bhargava extablished a method for representing the primitives of a pattern as nodes of a tree.

This paper includes an extension of (20) in several ways. Real patterns can indeed be more adequately represented by trees than by strings but there is still something missing. Branches of a tree can never intersect, i.e. paths down two distinct branches can never reach the same point, but, as we trace over various parts of a pattern, the same point can often be reached by several different routes from a given start point. To overcome this handicap (and others) we will not restrict ourselves just to trees but rather to tree-like structures called d-trees, where distinct branches can lead to the same node or nodes. In an actual pattern recognition scheme it is desirable to have the ability to construct the representative tree, or d-tree, using no outside information, once the set of primitives is known. The tree, or d-tree, can then be presented directly to a pseudoautomaton for acceptance or rejection, or possibly for classification among several different pattern classes. The method presented here has this ability.

The study of tree grammars, tree acceptors and their extensions as presented in this thesis also leads to some interesting results in the theory of formal languages. Some of these results are developed here in Chapters 3, 5 and 6, although Chapters 3 and 5 are more concerned with the practical problems associated with pattern recognition.

The work in this thesis also offers a contribution as it provides a step towards formalizing properties of graph transformations. The tree-like structures developed here are based on graphical properties rather than the functional definitions used in most other work (see Chapter 2). Although graphs in general are not treated here, d-trees (Definition 3.4) are a more general form of graphs than those previously studied by tree automata theorists. This work, which is another application of the study of multidimensional automata, constitutes a partial bridge from tree automata theory to a theory of graph automata.

1.2 APPLICATION AREAS OF MULTIDIMENSIONAL AUTOMATA

We will use the term multidimensional automata to denote mathematical models of machines which can accept, reject or classify inputs from spaces with dimensionality order greater than one. (One dimensional encodements of higher dimensional structures are also possible inputs.) A brief survey of several applications of multidimensional automata theory follows.

1.2.1 A LOGIC APPLICATION

Thatcher(40) presents a discussion and history of the question of decideability of the weak monadic second-order theory of multiple successors. The question was answered affirmatively for the one successor case using concepts from the area which has (later) become known as tree automata theory independently by Buchi(9) and Elgot(13). The problem was posed for the case of multiple successors by Buchi(9). Doner(12) was later able to generalize the tree theory methods of Buchi and Elgot and prove this result also.

1.2.2 INSIGHTS TO LANGUAGE THEORY

W.C. Rounds(36) shows how the newly developed theory of tree automata can be used to provide clearer insights into, and better proofs of, some of the classical theory of formal languages. One example is a simple proof showing that the class of context-free languages is closed under intersection with regular sets. New areas of investigation are opened also. For example, the theorem given in Peters and Ritchie(32), that the language analyzable[1] by a finite set of context-sensitive productions is context-free, is also shown in (36). (The proof originally given by Peters and Ritchie was obscure if not incorrect.)

Theorem 6.2 of this thesis provides another example of an application of tree automata theory for proving results in the area of formal language theory.

1.2.3 NATURAL LANGUAGE APPLICATIONS

N. Chomsky in (11) and in other works has shown that phrase structure grammars can not suffice to represent equivalence of meaning between such sentences as "Sincerity may frighten the boy." and "The boy may be frightened by sincerity.". Rather, he has proposed a more general type of generative system called a "transformational grammar". A transformational grammar combines string generating phrase structure rules with rules that provide transformations between trees representing the deep structure of such systems.

It has been observed by Aho and Ullman(2), by Martin and Vere(26) and by Rounds(35), that formalized tree transduction systems can provide

---

[1]"Analyzable by" essentially means "can be parsed using". For a more complete definition see (36).

a system for carrying out these transformations. Tree transducers are also considered as possible machines to aid in translations from one language to another. No practical work along these lines has yet been attempted but these research papers lay some of the theoretical foundations for it.

1.2.4 RECOGNIZING PATTERNS WITH MULTIDIMENSIONAL AUTOMATA

Automata which can accept or reject two-dimensional patterns are described by Blum and Hewitt(4) and by Savitch(37). One might think of this form of automaton as being a "bug" which is given a maze with filled and unfilled cells to wander around in. The bug then accepts or rejects the maze depending on whether the maze meets some pre-established criterion.

Some of the capabilities of the bug model given in (4) are that given a square maze it can decide if;

(1) The maze contains precisely k filled cells.

(2) The maze contains a single rectangle of filled cells.

(3) The maze contains a single square of filled cells with edges parallel to the maze boundaries.

Blum and Hewitt also show that bugs that can leave markers on cells, so that on future visits the cells can be recognized as previously visited, are more powerful than those that cannot leave markers.

The mazes considered in (37) are those with exactly two "doorways" from each cell leading to two other cells. Here a maze is accepted if there is a path of unfilled cells from a designated maze start cell to a designated maze goal cell. It is shown that constructing a bug to make such considerations is equivalent to constructing a Turing machine to accept or reject some coding of the maze.

Similar automata are used to provide an unusual characterization for the context-sensitive languages in Fischer(16).

Generally, pattern recognizing automata go hand-in-hand with pattern generating grammars. In an important paper by Pfaltz and Rosenfeld(33) generative schemes called web grammars are introduced. No accepting automata are defined; acceptance of a pattern generated by a web grammar can only be accomplished by parsing with the grammar.

<u>Example 1.1</u> A simple web grammar from Pfaltz and Rosenfeld(33).

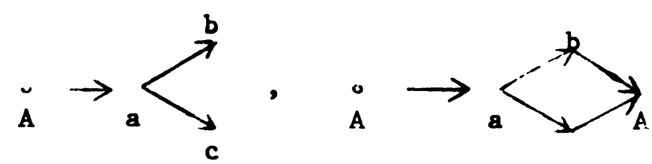
$G= \langle V_N, V_T, P, S \rangle$

$V_N=\{A\}$: Nonterminal symbols

$V_T=\{a,b,c\}$: Terminal symbols

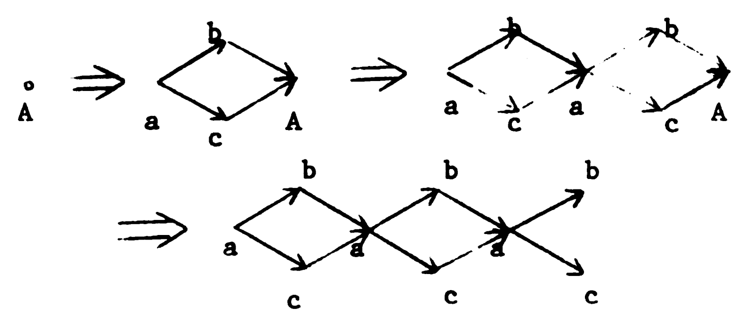
$S=\{A^\circ\}$: Start web

P= {



} : Productions

A derivation might be:



Any pattern, with only terminals labeling the joining of line segments, which can be generated by a web grammar is said to be in the pattern language. Pfaltz and Rosenfeld defined web grammars in such general terms, and allowed so many types of productions through the use of "embedding rules" (which describe exactly how each production is to be applied) that their system effectively subsumes any tree or graph production system that one might define. Unfortunately, however, it does not shed much light on the nature of such systems.

There is an essential difference in interpretation between the webs produced by the Pfaltz and Rosenfeld grammars and the multidimensional structures used by others (including this author). Web grammars produce structures which are themselves regarded as being patterns. On the other hand the strings, trees etc. used by others are regarded as partial representations for the pattern. Structures which represent patterns, rather than constitute patterns, provide an important step towards an abstraction of pattern recognition concepts.

The scheme developed by Fu and Bhargava (to be discussed in some detail in Chapter 3) offers the first application of multidimensional automata used for pattern recognition involving both a generative system and a machine to do the acceptance/rejection/classification. In previous cases this has been accomplished only through parsing. The material presented in Chapters 3, 4 and 5 of this dissertation provides a significant extension to this work; an extension that may lead to a practical recognition scheme for a number of pattern recognizing applications.

CHAPTER 2

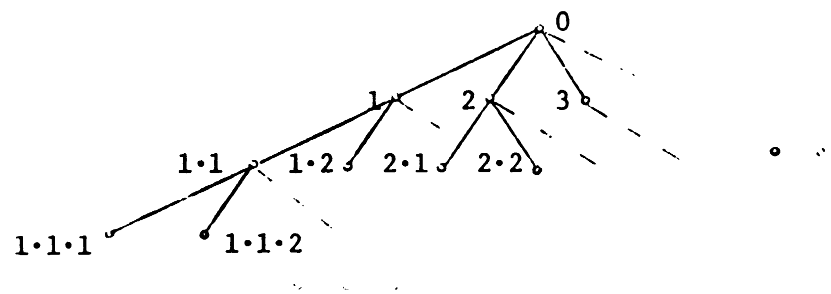
TREE AUTOMATA FORMALIZATION

It is difficult or impossible to discuss various aspects of tree automata theory such as subtrees, tree replacement, and tree grammars without an adequate formalism for the language of discussion. This chapter is an attempt to provide that framework and to introduce the principal results upon which the remainder of the dissertation will be based.

2.1 BASIC DEFINITIONS

Many definitions and previous results in tree automata theory will be used for our pattern recognizing automata. We begin with definitions leading towards the idea of a tree acceptor or pseudoautomaton. Most of the definitions come from Thatcher(39) and Brainerd(7) although appropriate notational changes have been made. A similar development can be found in Brainerd(7).

Definition 2.1 Let N^+ be the set of positive integers. Let U, the universal tree domain, be the free semi-group with identity, 0, generated by N^+ and a binary operation, $\cdot$. So, for all $a \in U$ we have $a \cdot 0 = 0 \cdot a = a$. U can be partially represented by the following figure:



Definition 2.2 The *depth* of $a \in U$ is denoted $d(a)$ and is defined as follows: $d(0)=0$, $d(a \cdot i)=d(a)+1$ for $i \in N^+$.

Definition 2.3 $a \leq b$ if $\exists\, x \in U \ni a \cdot x=b$. $a \geq b$ if $b \leq a$. Similarly $a<b$ if $a \leq b$ and $a \neq b$, and $a>b$ if $a \geq b$ and $a \neq b$. We say a and b are *incomparable* if $a \not\leq b$ and $b \not\leq a$.

Definition 2.4 D is a *tree domain* if:

(a) D is a finite subset of U,

(b) $b \in D$ and $a<b \Rightarrow a \in D$ and

(c) for $i,j \in N^+$, $a \cdot j \in D$ and $i<j \Rightarrow a \cdot i \in D$.

Definition 2.5 A *ranked alphabet* is a pair $\langle A,\sigma \rangle$ where A is a finite set of symbols and σ is a finite relation in $A \times N$.[1] (N=non-negative integers.) Let $A_n = \sigma^{-1}(n)$ so for all n, A_n will be a subset of A.

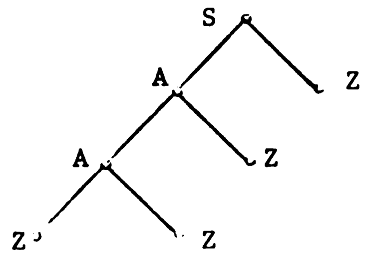
Definition 2.6 A *tree* (actually an *ordered tree*) over A (i.e. over $\langle A,\sigma \rangle$) is a function $\alpha : D \rightarrow A$ where D is a tree domain and for $a \in D$, $\max\{ i \mid a \cdot i \in D\} \in \sigma(\alpha(a))$. We denote the domain of a tree by $D(\alpha)$ or D_α. We will denote the set of all trees over Σ by T_Σ.

Example 2.1 Let $D_\alpha=\{0,1,2,1\cdot1,1\cdot2,1\cdot1\cdot1,1\cdot1\cdot2\}$, $\Sigma=\{ S,A,Z\}$, $\Sigma_0=\{Z\}$, $\Sigma_2=\{A,S\}$. The tree $\{(0,S),(1,A),(2,Z),(1\cdot1,A),(1\cdot2,Z),(1\cdot1\cdot1,Z),$

---

[1] In most previous work σ was assumed to be functional so each symbol had fixed rank. It was pointed out by Thatcher(43) that this is an unnecessary restriction and the theory holds for all finite σ.

(1·1·2,Z)} may be written as:



The following definitions allow us to deal with parts (subtrees) of trees.

<u>Definition 2.7</u> Let a,b,b' be members of U (the universal tree domain) such that a·b'=b. Then <u>b/a</u>=b' if a$\le$b,

<u>b/a</u> is undefined otherwise.

We now have b/0=b, a/a=0 and (if b/a is defined) a·(b/a)=(a·b)/a=b.

<u>Definition 2.8</u> Let α be a tree and "a" be a member of D . <u>α/a</u>= $\{(b,x)|(a\cdot b,x)\in\alpha\}$. α/a is called a subtree of α at a. Note that $\alpha\in T_A$ and a$\in$D $\Rightarrow \alpha/a \in T_A$.

2.2 STRING REPRESENTATIONS

Different authors use different methods to represent trees in string form. In this paper we will use what is commonly called pseudoterm notation[2] so the tree of Example 2.1 is represented by S(A(A(ZZ)Z)Z). Although pseudoterm notation is somewhat more awkward to write than other forms (for example postfix form) it has the advantage of always providing a unique representation for a tree regardless of the ranking relation defined whereas other methods only define a unique tree when used with the ranking relation. The notation used <u>does</u> have an effect

---

[2]This is also called bracketed notation, list notation or functional notation. (See Brainerd(6).)

on the definition of tree automata, but an equivalent formalization can be given for each notation.

Definition 2.9 The set τ_Σ of pseudoterms, usually just called terms, on Σ, is the smallest subset of $(\Sigma \cup \{(,)\})^*$ satisfying:[3]

(a) $\Sigma \subseteq \tau_\Sigma$

(b) If $n>0$ and $f \in \Sigma$ and $t_1, t_2, \cdots, t_n \in \tau_\Sigma$ then $f(t_1 t_2 \cdots t_n) \in \tau_\Sigma$.

We now define a recursive algorithm for writing the pseudoterm corresponding to a tree:

Algorithm 2.1

0. (We refer to the tree being operated upon as tree T.)
1. Write "a" where $(0,a) \in T$.
2. If $(1,a_1) \notin T$ then done: exit.
3. Write "(".
4. Write the pseudoterms (by calling Algorithm 2.1), in order, for subtrees T/1, T/2,$\cdots$,T/i where $i=\max\{j \mid (j,a_j) \in T\}$.
5. Write ")".
6. Done: exit.

After recursive applications of Algorithm 2.1 until an exit from the initial call occurs we will have the pseudoterm corresponding to the original tree.

2.3 TREE ACCEPTORS

We are now ready to define tree accepting machines. The definition used will essentially be Brainerd's(7) formalization of Thatcher's(39) definition.

---

[3]We assume that parentheses are not symbols of Σ.

Definition 2.10 Let $\langle A,\sigma \rangle$ be a ranked alphabet where $A=\{X_1,X_2,\cdots X_k\}$. A tree automaton, or pseudoautomaton, is a system $M=\langle Q,t_1,\cdots,t_k,F \rangle$ where:

(a) Q is a finite set of states.

(b) For each i, $1\leq i\leq k$, t_i is a relation in $Q^z \times Q$ for $(X_i,z)\in\sigma$.

(c) $F\subseteq Q$ is a set of final or accepting states.

If each t_i is a function $t_i:Q^z\rightarrow Q$ for all $(X_i,z)\in\sigma$ then M is deterministic, otherwise M is nondeterministic in which case we write $t_i(X_1,\cdots,X_n)\leadsto X_0$[4] iff $((X_1,\cdots,X_n),X_0)\in t_i$. If $(X_i,0)\in\sigma$ we write $t_i\leadsto X$ iff $(\lambda,X)\in t_i$.[5]

Notation: If $X=X_i\in A$, then t_X means t_i.

We now show how each automaton accepts or rejects a member of τ_A and thus defines a set of accepted trees.

Definition 2.11 The response, ρ, of a pseudoautomaton M to an input is defined as follows:

(a) If $x\in A_0$, $\rho(x)\leadsto X$ iff $t_x\leadsto X$.

(b) If $x\in A_n$, $n>0$, $\rho(x(\alpha_1\cdots\alpha_n))\leadsto X$ iff $\exists X_1,\cdots,X_n\in Q$, $t_x(X_1\cdots X_n)\leadsto X$ and $\rho(\alpha_i)\leadsto X_i$, $1\leq i\leq n$.

If M is deterministic ρ is a function with $\rho:\tau_A\rightarrow Q$ characterized by the following:

(a) If $x\in A_0$, $\rho(x)=t_x(\lambda)$.

(b) If $x\in A_n$, $n>0$, $\rho(x(\alpha_1\cdots\alpha_n))=t_x(\rho(\alpha_1)\cdots\rho(\alpha_n))$.

Definition 2.12 The behavior of a tree automaton $M=\langle Q,t_1,\cdots,t_n,F \rangle$ is $\{t|\rho(t)\in F\}$. This is also called the language of M, L(M).

---

[4]We may read symbol "$\leadsto$" as "goes to" or "contains".

[5]λ = the string of length 0.

<u>Example</u> <u>2.2</u> $A=\{0,1,X,S,B\}$, $A_0=\{0,1,X\}$, $A_1=\{B\}$, $A_3=\{S\}$. Consider the deterministic pseudoautomaton $M=\left\langle Q=\{1,0,S,B,X\},\ t_0,t_1,t_X,t_B,t_S,F=\{S\}\right\rangle$.

We define the t functions as: $t_0(\lambda)=0$

$t_1(\lambda)=1$

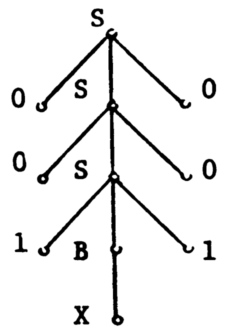
$t_X(\lambda)=X$

$t_B(X)=B$

$t_S(1B1)=S$

$t_S(0S0)=S$

We will find the response of M on tree:



This tree has pseudoterm representation: $S({}^{n}S(0S(1B(X)1)0)0)$.

Now $\rho(S(0S(0S(1B(X)1)0)0))=$

$t_S(\rho(0)\rho(S(0S(1B(X)1)0))\rho(0))=$

$t_S(t_0(\lambda)t_S(\rho(0)\rho(S(1B(X)1))\rho(0))t_0(\lambda))=$

$t_S(0t_S(t_0(\lambda)t_S(\rho(1)\rho(B(X))\rho(1))t_0(\lambda))0)=$

$t_S(0t_S(0t_S(t_1(\lambda)t_B(\rho(X))t_1(\lambda))0)0)=$

$t_S(0t_S(0t_S(1t_B(t_X(\lambda))1)0)0)=$

$t_S(0t_S(0t_S(1t_B(X)1)0)0)=$

$t_S(0t_S(0t_S(1B1)0)0)=$

$t_S(0t_S(0S0)0)=$

$t_S(0S0)=$

S

Since $S \in F$ the tree is accepted, i.e. this tree is an element of L(M). Actually we can show that M as constructed here will accept a tree, T, iff T is a parse tree from the context-free phrase structure grammar $G = \left\langle V_N=\{S,B\}, V_T=\{0,1,X\}, S, P=\{S\to 0S0, S\to 1B1, B\to X\}\right\rangle$. Note that $L(G)=\{0^n 1X10^n | n\geq 0\}$.

Although the procedure of recognizing a tree with a tree automaton seems to be a tedious task we will see later that it can be greatly simplified. We will be using tree automata as pattern recognition devices where a tree will represent a pattern to be accepted, rejected or perhaps classified by what final state the automaton ends in.

Definition 2.13 Given tree α, the frontier of α (abbreviated $fr(\alpha)$) is the string defined recursively as:

(a) If $\alpha=(0,x)$ (i.e. α is a single node tree) $fr(\alpha)=x$.

(b) Otherwise, $fr(\alpha)=fr(\alpha/1)fr(\alpha/2)\cdots fr(\alpha/n)$ for $n=\max\{i | i\in N^+, i\in D_\alpha\}$.

For example, the frontier of the tree in Example 2.1 is ZZZZ. The frontier of the tree in Example 2.2 is 001X100.

Theorem 2.1 (Thatcher(39)) A set of trees, T, is the behavior of a tree automaton iff T is a projection[6] of the set of derivation trees generated by some context-free grammar.

Corollary 2.1 The frontier of the behavior of a tree automaton is a context-free language.

2.4 TREE GRAMMARS

Definition 2.14 Let $a\in D_\alpha$, $\alpha,\beta\in T_A$. $\alpha(a\leftarrow\beta)=\{(b,x)\in\alpha | b\not\geq a\}\cup\{(a\cdot b.x) | (b,x)\in\beta\}$.

---

[6] A projection is a function from the (finite) alphabet of the context-free grammar onto a finite alphabet. It is known that the class of context-free languages is closed under projections(see (22)).

This is the result of replacing the subtree α/a at "a" by the tree β.

Definition 2.15 (Brainerd(7)) A regular tree grammar[7] over A (i.e. over ranked alphabet $\langle A,\sigma\rangle$) is a system $G=\langle B,\sigma',P,\Gamma\rangle$ satisfying:

(a) $\langle B,\sigma'\rangle$ is a finite ranked alphabet such that $A\subseteq B$ and $\sigma'|A=\sigma$ (i.e. relation σ' over elements of A is exactly equivalent to σ). The elements of A and B-A are called terminal and non-terminal symbols respectively.[8]

(b) P is a finite set of production rules of the form $\Phi\rightarrow\Psi$ where $\Phi,\Psi\in T_B$.

(c) $\Gamma\subseteq T_B$ is a finite set of axioms (the start trees).

The following definition indicated how a regular tree grammar generates trees.

Definition 2.16 (Brainerd(7)) $\alpha \overset{a}{\Rightarrow} \beta$ is in G iff $\exists$ a rule $\Phi\rightarrow\Psi$ in P such that $\alpha/a=\Phi$ and $\beta=\alpha(a\leftarrow\Psi)$. $\alpha\Rightarrow\beta$ is in G iff $\exists\, a\in D_\alpha$ with $\alpha \overset{a}{\Rightarrow} \beta$. $\alpha \overset{*}{\Rightarrow} \beta$ is in G iff $\exists\, \alpha_0,\cdots,\alpha_m$, $m\geq 0$ such that $\alpha=\alpha_0\Rightarrow\alpha_1\Rightarrow\cdots\Rightarrow\alpha_m=\beta$ is in G. The sequence $\alpha_0,\cdots,\alpha_m$ is called a derivation or deduction of β from α, and m is the length of the derivation. We will think of a regular tree grammar over $\langle\Sigma,\sigma\rangle$ as generating trees in T_Σ.

Definition 2.17 If G is a regular tree grammar over A, then $L(G)=\{\alpha\in T_A \mid \exists\,\gamma\in\Gamma$ such that $\gamma \overset{*}{\Rightarrow} \alpha$ is in G$\}$ is the set of trees generated by G. Regular tree grammars G and G' are equivalent if $L(G)=L(G')$.

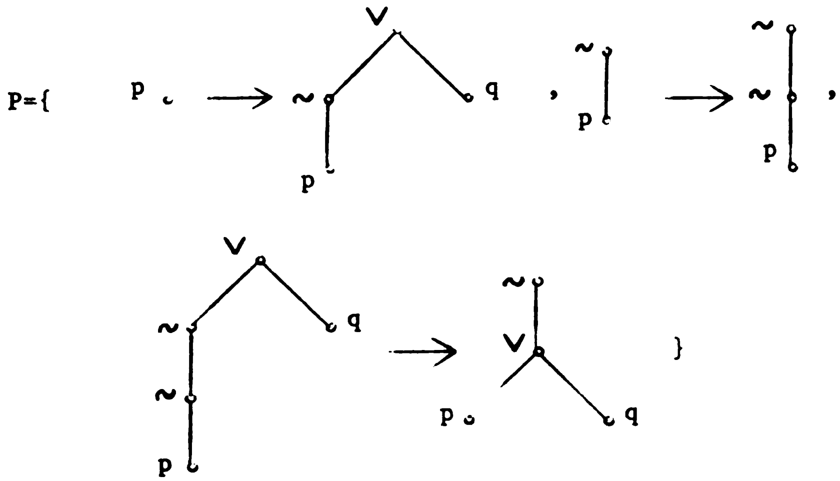
---

[7]This was originally called a regular system.

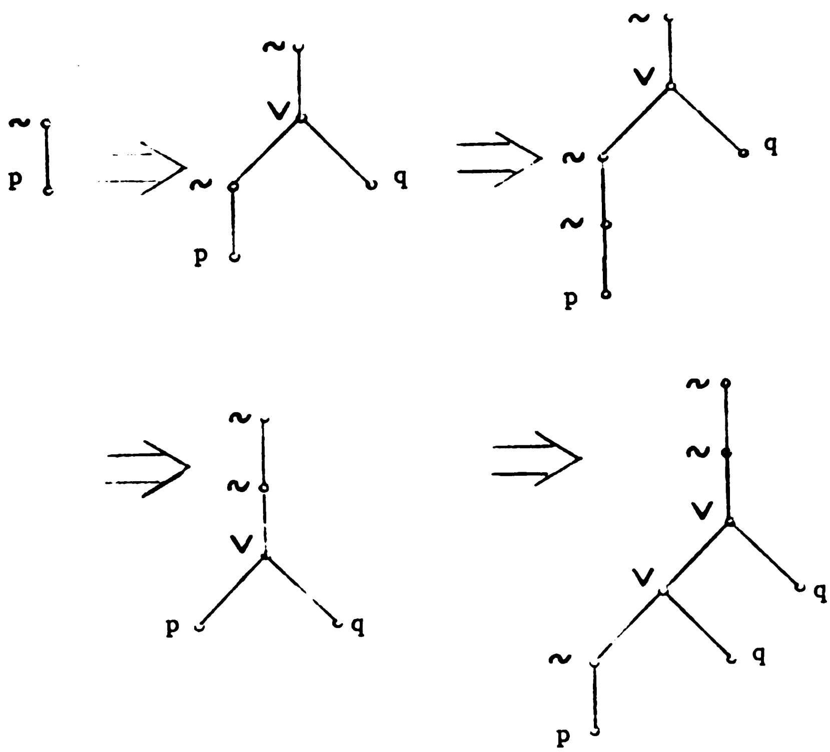
[8]The terms terminal and nonterminal are used in a somewhat different sense than one may be accustomed to. Trees in the language of a tree grammar will have all their nodes labeled with terminal symbols but it is permissible to have tree productions whose left hand sides use terminals and whose right hand sides use nonterminals.

<u>Example</u> <u>2.3</u> (Brainerd(7)) Let $S = \langle A,\sigma,P,\Gamma \rangle$. $A_0=\{p,q\}$, $A_1=\{\sim\}$, $A_2=\{\vee\}$.

$\Gamma=\{$  $\}$.



This grammar has no nonterminal symbols. One derivation is:



Theorem 2.2 (Single Axiom Theorem, Brainerd(7)) For every regular tree grammar $G=\langle B,\sigma,P,\Gamma\rangle$ one can effectively construct an equivalent regular tree grammar $G'=\langle B',\sigma',P',\Gamma'\rangle$ such that Γ' consists of a single nonterminal symbol, i.e. a tree of the form $\{(0,z)\}$, $z\in B_0$.

Definition 2.18 A tree grammar $G=\langle B,\sigma,P,Z\rangle$ over A is expansive[9] if each rule in P is of the form $X_0\to x(X_1\cdots X_n)$ where $x\in A_n$ and $X_0,X_1,\cdots,X_n\in B-A$, or of the form $X_0\to x$ where $x\in A_0$.

Theorem 2.3 (Brainerd(7)) For each regular tree grammar $G=\langle B,r,P,S\rangle$ over Σ one can effectively construct an equivalent expansive grammar with a single axiom.

Theorem 2.4 (Thatcher(40)) A set of trees, T, is accepted by a nondeterministic tree automaton iff T is accepted by a deterministic automaton. Furthermore, given a nondeterministic tree automaton one can effectively construct an equivalent deterministic tree automaton.

Theorem 2.4 is shown through a subset construction similar to that used to construct deterministic finite state machines.

The following result is basic to our later pattern recognition as it allows us to build an acceptor for trees (and later, tree-like structures) that represent patterns from a given set.

Theorem 2.5 (Brainerd(7)) For every regular tree grammar, G, one can effectively construct a deterministic tree automaton, M, such that $L(M)=L(G)$.

---

[9] Intuitively an expansive grammar is one where at each application of a production the generated tree grows or expands, it can never contract.

For regular tree grammar $G=\langle B,r',P,S\rangle$ over $\langle \Sigma=\{x_1,x_2,\cdots,x_k\},r\rangle$ the construction procedure may be summarized as follows:

Step(1) Obtain an equivalent expansive regular tree grammar $\langle B',r',P',S'\rangle$ with single node trees as axioms.

Step(2) The equivalent nondeterministic tree automaton is $M=\langle (B'-\Sigma)\cup S'',t_1,t_2,\cdots,t_k,S''\rangle$ where $t_x(X_1\cdots X_n)\sim X_0$ iff $X_0\rightarrow x(X_1\cdots X_n)$ is a rule in P', and S" is the set containing precisely the labels of start trees in S'.

Step(3) Now construct a deterministic tree automaton equivalent to M.

Brainerd has also shown the converse of Theorem 2.5:

Theorem 2.6 For every tree automaton, M, one can effectively construct a regular tree grammar, G, such that L(M)=L(G).

Taking Theorems 2.5 and 2.6 together we have:

Theorem 2.7 The sets of trees generated by regular tree grammars are exactly those accepted by finite tree automata.

CHAPTER 3

UNORDERED TREES

3.1 MOTIVATIONS

A general theory of graph grammars, graph transducers and graph automata might be viewed as one goal of research of this type. The results presented in this chapter provide a partial bridge between tree automata theory and this goal. Grammars and acceptors for certain classes of digraphs are shown with a number of interesting results. The digraphs are, however, still restricted in important ways.

On a more immediate level, the work done by Fu and Bhargava(20) who first used tree automata theory in pattern recognition was the original stimulus. The following example of theirs will serve both to illustrate the methodology and to point the way towards necessary extensions. In this paper, we will view patterns as connected figures which are built up of more basic figures called primitives (see Chapter 4.) The object of this research is not to aid in finding the primitives; rather we assume we have the primitives (or at least have the portions of patterns which may be regarded as primitives) and want to work from them while attempting to identify the patterns.

Example 3.1 (Fu and Bhargava(20)) We want to recognize "squares on top of and to the right of squares". A better pattern class definition might be to say that we want to recognize all patterns which can be generated by the following procedure:

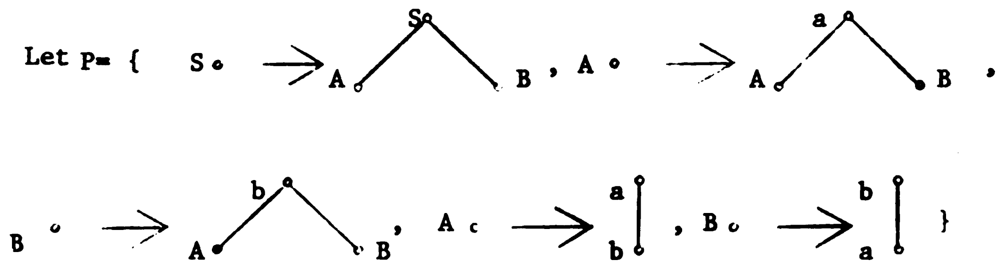
(a) Draw a square with vertical and horizontal sides.

(b) Halt or draw a square immediately to the right of or above an existing square.

(c) Return to step(b).

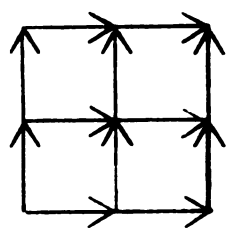
Let regular tree grammar G= $\langle B, r, P, S\rangle$ where B={S,a,b,A,B} over Σ= {S,a,b}. B_0={a,b}, B_1={a,b}, B_2={a,b,S}. a = →, b = ↑, length(a)= length(b).

Let P= {

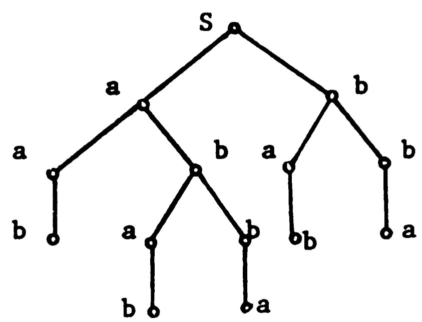


}

Given figure:



the following tree (which can be derived using G) represents it:



This tree's pseudoterm representation is S(a(a(b)b(a(b)b(a)))b(a(b)b(a))). The tree automaton which accepts trees generated by G is $M=\left\langle\{A,B\}, t_a,t_b,t_S,\{S\}\right\rangle$ where the t functions are defined by:

$$t_S(AB)=S$$

$$t_a(AB)=A$$

$$t_b(AB)=B$$

$$t_a(b)=A$$

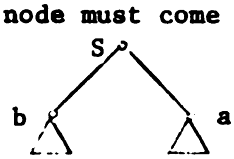
$$t_b(a)=B$$

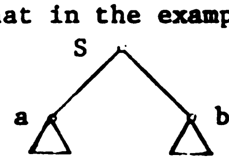
$$t_b(\lambda)=b$$

$$t_a(\lambda)=a$$

Notice that the nonterminals of the regular tree grammar have become the states of the accepting tree automaton. It will be worthwile for the interested reader to follow the steps through as M accepts some short derivation from G, i.e. find the response, ρ, to string, t, which represents (in pseudoterm form) a tree generated by G.

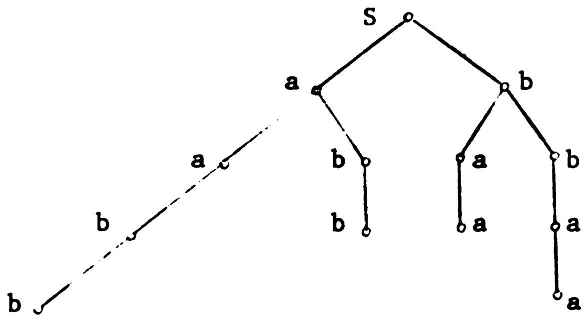
Given a pattern and its primitives we would like to be able to generate (and later accept) trees (or tree-like structures) in a manner similar to that of Fu and Bhargava but with extensions meeting the following criteria:

(1) When constructing a structure to represent a pattern we do not want to have to use any prior information concerning which of the descendants of a node must come first - note that in the example if we had generated 

rather than 

we would not have been able to accept the tree with our automaton. We do not want to continually have to check on this kind of order information.

(2) We want to be able to construct our structure without the necessity of referring back to the productions in the grammar to find out which new primitives must be adjoined to which current ones. In the example when constructing the representative tree, how did we know that we could not start at S and construct the following tree (which does represent the figure but will not be accepted by M)?



The only way we know not to make the tree in this way is by referring back to the production rules and observing that this is not an acceptable tree. In fact, if we can exhaust our supply of primitives for a pattern by constructing a tree where we have consulted a production rule at each stage to see if we are making a proper expansion we do not need to present the tree to an acceptor at all - it must be acceptable because of the way it was constructed.

3.2 GRAPH THEORY BASES

The considerations of the previous section lead us to the following development with graph theoretical definitions. Graph definitions are from Harary(21).

Definition 3.1 A digraph, D, consists of a finite, nonempty set of points or nodes, U, and a set of ordered pairs of distinct points in U,

V. We may write D= $\langle U,V \rangle$. Any pair (u_1,u_2) in V is called an arc or directed edge. Arc (u_1,u_2) may be shown as an arrow from point u_1 to point u_2.

Definition 3.2 A walk in a digraph is an alternating sequence of points and arcs $u_0,x_1,u_1,x_2,\cdots,x_n,u_n$ in which each arc x_i is (u_{i-1},u_i). The length of a walk is the number of occurences of arcs in it. A path is a walk in which all points are distinct. A closed walk is a walk with the same first and last points and a cycle is a nontrivial closed walk with all points distinct (except the first and last). A digraph is acyclic if it contains no cycles.

Definition 3.3[1] A digraph, D, is rooted if it contains a special point, called the root, r, such that D contains a walk from r to each other point in D. A digraph is labeled when each node is assigned a unique name as a label.

Definition 3.4 A d-tree is a rooted, acyclic, labeled digraph. The labels used for d-trees will be ordered pairs, a:x. We call "a" the first label coordinate, "x" the second. In practice it will be the "x" which serves to uniquely identify different nodes.

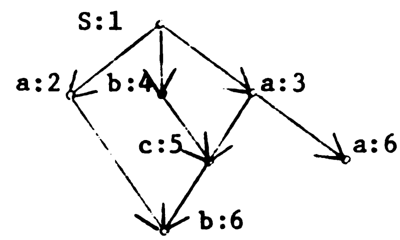
When writing d-tree D we will assume that node $a{:}x_a$ physically shown higher than node $b{:}x_b$ and line segment $a{:}x_a$-$b{:}x_b$ shown means that there is an arc $(a{:}x_a,b{:}x_b)$ in D. We are taking advantage of the

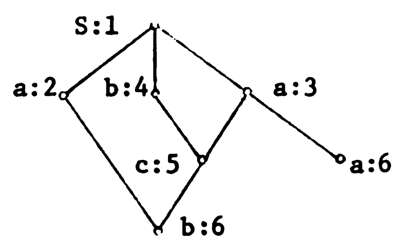
---

[1]Chapter 2 and the first part of this chapter have consisted primarily of results and definitions due to other researchers. From this point on, the definitions and results are original with this author unless otherwise specified. The work, however, does build on the work of others, most particularly Thatcher (39,42) and Brainerd(7).

fact that a d-tree has an implied partial ordering on its nodes with a least element, the root.

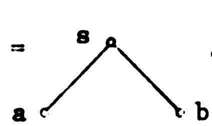
Example 3.2

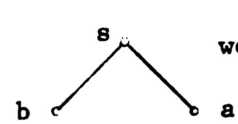
D-tree T = 

can be written as T = 

We define unordered tree differently than other authors (see Knuth (23) and Harary(21)).

Definition 3.5 A u-tree or unordered tree is a rooted digraph with each node assigned a (not necessarily unique) label and with exactly one path from the root to each node. We denote the set of all u-trees with labels from alphabet, B, as $u_T B$. The same convention will be used to represent arcs of u-trees without arrows as that used for d-trees.

The concept of order is the essential difference between trees and u-trees. Given trees A = 

and B = 

we

would in no way want to call them the same but if we regard A and B as representations of u-trees they are just different representations for the same u-tree.

Definition 3.6 If a d-tree or u-tree contains arc (a,b), node "a" will be called a predecessor of "b" and node "b" will be a successor of "a". The set frontier of a d-tree or u-tree is the set of all nodes with no successors.

Notation: Where there is no chance for confusion we may denote a node with label x as node x.

Definition 3.7 A root-to-frontier path in a d-tree or u-tree is a path whose first point is the root and whose last point is an element of the set frontier. The depth of a u-tree, t, is the max $\{n|n=\text{length of a root-to-frontier path in } r\}$.

Definition 3.8 We define the natural correspondence of u-trees to d-trees through the following algorithm, Algorithm 3.1. If u-tree U is the result of applying Algorithm 3.1 to d-tree D then we say U is the naturally corresponding u-tree to D. Informally we construct U by "unfolding" the non-unique root-to-frontier paths in D to give unique paths in U.

In this algorithm we use two pushdown stacks, P and S. P and S will point to distinct nodes of d-tree D and u-tree U respectively. Our notation will be: P=x means that the top of stack P points to node s and P←x means to push down P by placing x on it. We also have a method of marking nodes of D and later erasing our marks. The algorithm attempts to simulate the steps a person might follow as he proceeds through D and U; P and S give him the ability to "back up" a root-to-frontier path and the marking allows him to see what has already been expanded.

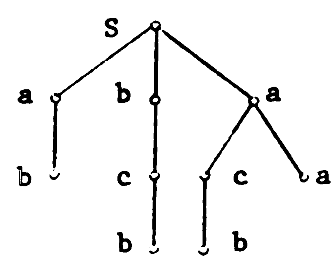
<u>Algorithm 3.1</u> (Algorithm to define and construct the naturally corresponding u-tree U, to d-tree D.)

Initial conditions: All nodes of D are unmarked. P=r for r the root of D. S=0.

0. Denote current node as node P of D with label a:x.
1. Construct new node P' of U with label "a". If S=0 make node P' the root of U. If S≠0 make node P' a successor of node S.
2. Mark node P. S←P'.
3. If node P has no unmarked successors then erase the mark from each marked successor (if any), pop S, pop P and go to step 4. Else go to step 5.
4. If S=0 then done: exit. Else go to step 3.
5. Pick an unmarked successor of P, say node M. P←M. Go to step 0.

<u>Example 3.3</u> A representation of the naturally corresponding u-tree to d-tree T of Example 3.2 is:

T' =



<u>Definition 3.9</u> U-tree $A_1 = \langle U_1, V_1 \rangle$ is a <u>sub-u-tree</u> of u-tree $A_2 = \langle U_2, V_2 \rangle$ if:

(a) $U_1 \subseteq U_2$ such that for all $p_1 \in U_1$ if $(p_1, p_2) \in V_2 \Rightarrow p_2 \in U_1$, furthermore for $p \in U_1$, label(p)=x in A_1 iff label(p)=x in A_2.

(b) V_1 consists of exactly those pairs in V_2 whose components are both in U_1.

Definition 3.10 U-tree A_1 is identical to u-tree A_2 if each is a sub-u-tree of the other, and we may say $A_1 = A_2$.

3.3 STRING REPRESENTATIONS AND THE IMPLIED RELATIONSHIPS

We want to be able to represent d-trees and u-trees in linear form, as pseudoterms (see Definition 2.9). The correspondence between u-trees and their pseudoterms is given by:

(a) For the single node t, the corresponding term is t.

(b) For arcs $(t_0,t_1),(t_0,t_2),\cdots,(t_0,t_n)$ being the only arcs with first coordinate t_0, with $f_1,f_2,\cdots,f_n$ respectively being the terms corresponding to $t_1,t_2,\cdots,t_n$, the set of corresponding terms is $\{t_0(Y)|Y$ is a permutation of $\{f_1,f_2,\cdots,f_n\}\}$.

The correspondence between d-trees and their pseudoterms is given by:

(a) For the single node t:x the corresponding term is t.

(b) For arcs $(t_0{:}x_0,t_1{:}x_1),(t_0{:}x_0,t_2{:}x_2),\cdots,(t_0{:}x_0,t_n{:}x_n)$ being the only arcs with first coordinate $t_0{:}x_0$ with $f_1,f_2,\cdots,f_n$ respectively being the terms corresponding to $t_1{:}x_1,t_2{:}x_2,\cdots,$ $t_n{:}x_n$ the set of corresponding terms is $\{t_0(Y)|Y$ is a permutation of $\{f_1,\cdots,f_n\}\}$.

A corresponding pseudoterm for the d-tree in Example 3.2 and the u-tree in Example 3.3 is S(a(b)b(c(b))a(c(b)a)). Note that if B is the corresponding u-tree to d-tree A, both B and A will have the same pseudoterm representations. Again we stress the fact that neither the representation in Example 3.2 nor that in Example 3.3 is unique: there are several more ways of writing each of these structures. A tree can serve as a representation for a u-tree; we can regard T' of Example 3.3

as being a tree which is one representation for the u-tree.

In order to firmly establish the relationships that exist between d-trees, u-trees, trees and pseudoterms we will define the following mappings. (Here we distinguish between "mapping" and "function" by allowing a mapping from set A to set B to be any relation on A×B such that each element of A will be related to one or more elements of B whereas a function will be a relation on A×B such that each element of A will be related to exactly one element of B.)

Define c_1: d-trees→u-trees by the natural correspondence.

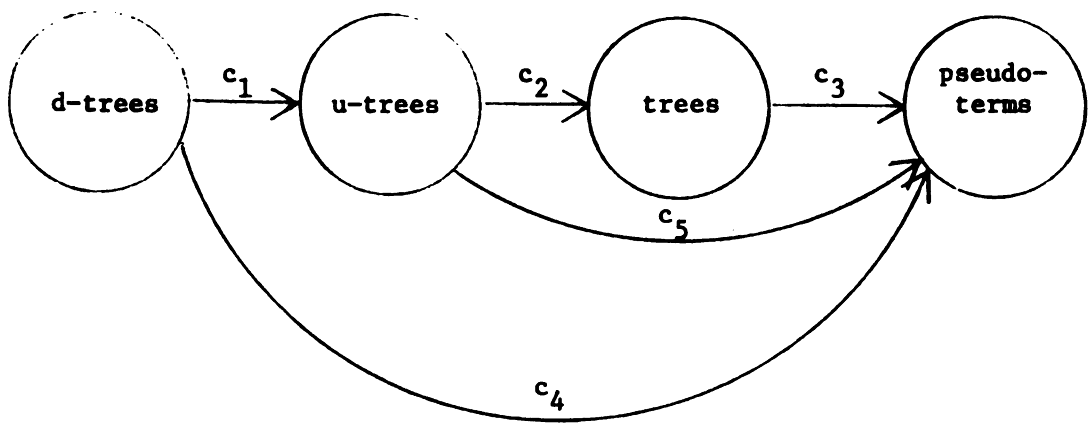
Define c_2: u-trees→trees by $t \in c_2(u)$ iff t is a tree which can serve as one representation of u-tree u.

Define c_3: trees→pseudoterms by Algorithm 2.1.

Define c_4: d-trees→pseudoterms by the above correspondence.

Define c_5: u-trees→pseudoterms by the above correspondence.

The following digraph illustrates these mappings:



All d-trees, u-trees and trees are over some ranked alphabet Σ.

Pseudoterms are over $\Sigma \cup \{),(\}$.

We may make the following observations concerning these mappings:

c_1 is a function since each d-tree has a unique naturally corresponding u-tree.

c_2 is not a function. For example $c_2({}_b\overset{a}{\wedge}{}_c) = \{ {}_b\overset{a}{\wedge}{}_c, {}_c\overset{a}{\wedge}{}_b \}$.

c_3 is a function. (See Algorithm 2.1.)

c_4 is not a function. Note that $c_4(d)=c_3(c_2(c_1(d)))$ which (because of the inclusion of c_2) is not a function.

c_5 is not a function. Note that $c_5(u)=c_3(c_2(u))$ which is not a function.

c_1^{-1} is not a function since[2] $c_1(a{:}x)=c_1(a{:}y)=a$.

c_2^{-1} is a function since given a tree it can serve as a representation of exactly one u-tree.

c_3^{-1} is a function. (See Section 2.2.)

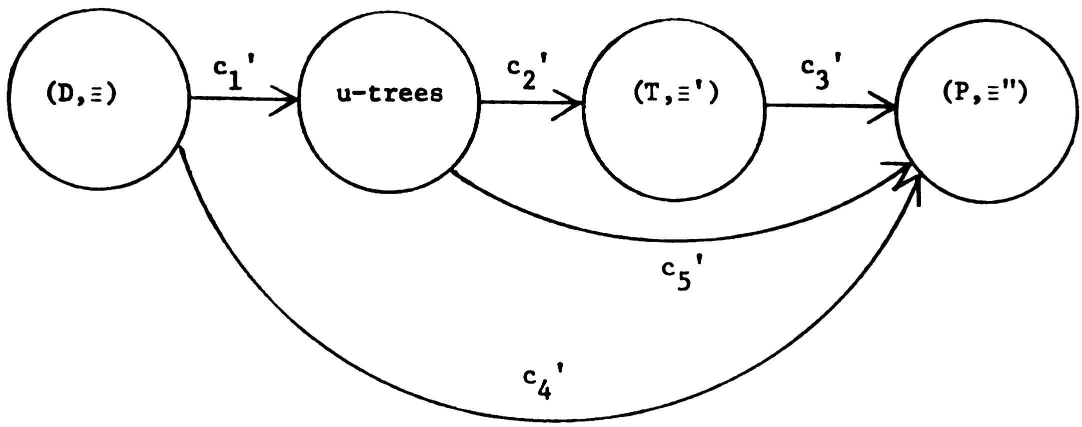
c_4^{-1} is not a function. We note that $c_4^{-1}(p)=c_1^{-1}(c_2^{-1}(c_3^{-1}(p)))$ which (because of the inclusion of c_1^{-1}) is not a function.

c_5^{-1} is a function. We note that $c_5^{-1}(p)=c_2^{-1}(c_3^{-1}(p))$.

We now define some equivalence relations based on the above functions. Let $(D,\equiv)$ be the equivalence relation on d-trees with $d_1 \equiv d_2$ iff $c_1(d_1)=c_1(d_2)$. Let $(T,\equiv')$ be the equivalence relation defined on trees by $t_1 \equiv' t_2$ iff $c_2^{-1}(t_1)=c_2^{-1}(t_2)$. Let $(P,\equiv'')$ be the equivalence relation defined on pseudoterms by $p_1 \equiv'' p_2$ iff $c_3^{-1}(p_1) \equiv' c_3^{-1}(p_2)$. Define $c_1',\cdots,c_5'$ in a similar manner to $c_1,\cdots,c_5$ except they are now defined on equivalence classes where appropriate, giving the following digraph:

---

[2]Notation: Regard (a:x) as the single node d-tree with label a:x. Similarly for (a:y) and a.



It is clear that $c_1', \cdots, c_5'$ are functions (as are $c_1'^{-1}, \cdots, c_5'^{-1}$) so each of $c_1', \cdots, c_5'$ must be one-one. Each equivalence class in $(D, \equiv)$ will be infinite but each equivalence class in $(T, \equiv')$ and in $(P, \equiv'')$ will be finite.

These mappings and equivalence classes will be used as we develop the theory of unordered tree automata. Two things are perhaps most important to observe:

(1) Given either a u-tree, a tree or a pseudoterm we can immediately and uniquely determine the appropriate corresponding pair among u-trees, classes in $(T, \equiv')$ and classes in $(P, \equiv'')$.

(2) Given a d-tree, it immediately defines a class of pseudoterms which correspond to a unique u-tree.

3.4 U-TREE GRAMMARS AND AUTOMATA

Definition 3.11 A regular u-tree grammar, abbreviated RUTG, over ranked alphabet $\langle A, \sigma \rangle$ is a system $M = \langle B, \sigma', P, \Gamma \rangle$ satisfying:

(a) $\langle B, \sigma' \rangle$ is a finite ranked alphabet such that $A \subseteq B$ and $\sigma' | A = \sigma$.

(b) P is a finite set of production rules of the form $\Phi \rightarrow \Psi$ where $\Phi, \Psi \in u_T B$.

(c) $\Gamma \subseteq u_T B$ is a finite set of <u>axioms</u>.

(d) For $t \in \Gamma$ or for t which is either the LHS or RHS[3] of a production in P, if t has exactly n arcs with first coordinate c, then $c \in A_n$.

We will think of a RUTG over $\langle A,\sigma \rangle$ as generating u-trees with labels in A. The following indicates how a RUTG generates u-trees:

Let α,β,ϕ,ψ be u-trees over B such that $\phi \mapsto \psi$ and $\alpha = \langle U_\alpha, V_\alpha \rangle$, $\beta = \langle U_\beta, V_\beta \rangle$, $\phi = \langle U_\phi, V_\phi \rangle$, $\psi = \langle U_\psi, V_\psi \rangle$. We say <u>$\alpha \Rightarrow \beta$</u> if $\exists$ a sub-u-tree of α, say $S = \langle U_S, V_S \rangle$, such that {paths in S}={paths in ϕ} and if $U_\beta = (U_\alpha - U_S) \cup U_\psi$ and $V_\beta = (V\alpha - (V_S \cup \{(u_i, u_{rS})\})) \cup V_\psi \cup \{(u_i, u_{r\psi})\}$ where (u_i, u_{rS}) is the arc that connected a node of U_α, node u_i, to the root, u_{rS}, of S and $(u_i, u_{r\psi})$ is a new arc connecting u_i to the root, $u_{r\psi}$, of ψ. We say <u>$\alpha \overset{*}{\Rightarrow} \beta$</u> if $\exists$ a sequence of u-trees $\alpha_0, \alpha_1, \cdots, \alpha_n$ such that $\alpha = \alpha_0 \Rightarrow \cdots \Rightarrow \alpha_n = \beta$. Given regular u-tree grammar $G = \langle B, \sigma', P, \Gamma \rangle$ over $\langle A, \sigma \rangle$, the <u>language</u> of G, L(G), is the set, T, of u-trees over A such that $t \in T$ iff $\gamma \overset{*}{\Rightarrow} t$ for some $\gamma \in \Gamma$.

<u>Definition 3.12</u> RUTG S is <u>equivalent</u> to RUTG S' if L(S)=L(S').

<u>Definition 3.13</u> Given set A, we define a set of equivalence classes, $(A^n, \equiv)$, on the set of strings of length n over A by letting $x \equiv y$ iff x is a permutation of the symbols in y for $x,y \in A^n$. We denote an individual class in $(A^n, \equiv)$ by overscoring any individual permutation in the class. Example: $\overline{ab}$ means the class of permutations on string "ab". This is {ab,ba}.

---

[3]LHS = left hand side, RHS = right hand side.

<u>Definition 3.14</u> Let $\langle A,\sigma\rangle$ be a ranked alphabet where $A=\{X_1,X_2,\cdots,X_k\}$. A <u>u-tree automaton</u>, abbreviated UTA, is a system $B=\langle Q,t_1,\cdots,t_k,F\rangle$ where:

(a) Q is a finite set of states.

(b) For each i, $1\le i\le k$, t_i is a relation in $(Q^z,\equiv)\times Q$ for $(X_i,z)\in\sigma$.

(c) $F\subseteq Q$ is a set of <u>final</u> or <u>accepting</u> states.

<u>Definition 3.15</u> If each t_i is a function $t_i:Q^z\to Q$ for all $(X_i,z)\in\sigma$ then M is <u>deterministic</u>, otherwise M is <u>nondeterministic</u> in which case we write $t_i(\overline{X_1\cdots X_n})\sim X_0$ iff $((\overline{X_1\cdots X_n}),X_0)\in t_i$. If $(X_i,0)\in\sigma$ we write $t_i\sim X$ iff $(\lambda,X)\in t_i$.

<u>Notation</u>: If $X=X_i\in A$, we will often write t_X meaning t_i.

<u>Definition 3.16</u> We now show how each automaton accepts or rejects a member of τ_A and thus defines a set of accepted trees: The <u>response</u>, ρ, of UTA M to an input is defined as follows:

(a) If $x\in A_0$, $\rho(x)\sim X$ iff $t_x\sim X$.

(b) If the input is $\alpha=\langle U,V\rangle$ which has pseudoterm representation $x(\alpha_1\alpha_2\cdots\alpha_n)$, $x\in A_n$ for $n>0$, where $\{\alpha_i|1\le i\le n\}$ contains the pseudoterm representations for sub-u-trees of $\alpha \ni (x,\alpha_i)\in V$ for $1\le i\le n$ then $\rho(\alpha)\sim X$ iff $\exists\, X_1,\cdots,X_n\in Q$ such that $t_x(\overline{X_1\cdots X_n})\sim X$ and $\rho(\alpha_i)\sim X_i$, $1\le i\le n$.

If M is deterministic, ρ is a function such that $\rho:\tau_A\to Q$ characterized by the following:

(a) If $x\in A_0$, $\rho(x)=t_x(\lambda)$.

(b) If the input is as in (b) for nondeterministic u-tree automata then $\rho(\alpha)=t_x(\overline{\rho(\alpha_1)\cdots\rho(\alpha_n)})$.

<u>Definition 3.17</u> Given UTA $M=\langle Q,t_1,\cdots,t_n,F\rangle$ the <u>behavior</u> of M is $\{t|\rho(t)\in F\}$. This is also called the <u>language</u> of M, L(M).

We now proceed to establish that RUTGs have the same properties in relation to generated u-trees that regular tree grammars (henceforth abbreviated RTGs) do for trees. Furthermore RUTGs bear the same relationship to UTAs that RTGs do to tree automata (henceforth abbreviated TAs). The development presented herein for unordered trees is similar to Brainerd's development for ordered trees.

Definition 3.18 RUTG $G=\langle B,\sigma,P,\Gamma\rangle$ is reduced if Γ consists of exactly one single-node start tree.

Definition 3.19 RUTG $G=\langle B,\sigma,P,\Gamma\rangle$ over A is expansive if each rule in P is of the form $X_0\to x(\overline{X_1\cdots X_n})$ where $x\in A_n$ and $X_0,X_1,\cdots,X_n\in B-A$ or of the form $X_0\to x$ where $x\in A_0$.

Lemma 3.1 For each RUTG $G=\langle B,\sigma,P,S\rangle$ over Σ one can effectively construct an equivalent reduced expansive grammar.

Proof WOLOG assume $P=\{\phi_i\to\psi_i \mid 1\le i\le r\}$ and $S=\{\alpha_j \mid 1\le j\le k\}$.[4] Referring to function c_2' of Section 3.3 we see that for all $u\in$ {u-trees over Σ} $c_2'(u)\ne\phi$. Therefore for all ϕ_i,ψ_i,α_j we can pick t^i_ϕ, t^i_ψ and t^j_α respectively such that $t^i_\phi\in c_2'(\phi_i)$, $t^i_\psi\in c_2'(\psi_i)$ and $t^j_\alpha=c_2'(\alpha_j)$. We now construct RTG $G'=\langle B',\sigma',P',S'\rangle \ni B'=B$, $\sigma'=\sigma$, $P'=\{t^i_\phi\to t^i_\psi \mid 1\le i\le r\}$ and $S'=\{t^j_\alpha \mid 1\le j\le k\}$. Now for all $t\in L(G')$ we know that $t\in c_2'(u)$ for some $u\in L(G)$ and furthermore for all $u\in L(G)$ $\exists$ $t\in L(G')$ such that $t\in c_2'(u)$. (G' is a "microcosm" of G; it functions similarly but at each application of a production it produces one representation of the u-tree that G produces.) We know by Theorem 2.3 that we can construct RTG $G''=\langle B'',\sigma'',P'',S''\rangle \ni G''$ is reduced and expansive and $L(G'')=L(G')$, therefore we can make the same

---

[4]By writing symbol α_j we mean the single node u-tree with label α_j.

observations about G" as about G', namely that given $t \in L(G'') \Rightarrow t \in c_2'(u)$ for some $u \in L(G)$ and furthermore given $u \in L(G) \Rightarrow \exists\, t \in L(G'')$ such that $t \in c_2'(u)$. (G" may, however, produce these terminal trees by a completely different procedure than G or G'.) We now define RUTG $G_f = \langle B_f, \sigma_f . P_f, S_f \rangle$ such that $B_f = B''$, $\sigma_f = \sigma''$, $S_f = S''$ and $P_f = \{X_0 \to x \mid X_0 \to x \in P''\} \cup \{X_0 \to x(\overline{X_1 \cdots X_n}) \mid X_0 \to x(X_1 \cdots X_n) \in P''\}$.

So RUTG G_f is clearly reduced and expansive and furthermore $L(G_f) = L(G)$. Q.E.D.

<u>Lemma</u> <u>3.2</u> For each reduced expansive RUTG $S = \langle B, \sigma, P, Z \rangle$ over A, one can effectively construct a nondeterministic UTA M, such that $L(M) = L(S)$.

<u>Proof</u> Let $M = \langle Q = (B-A) \cup \{Z\}, t_1, \cdots, t_k, \{Z\} \rangle$ where the move functions are $t_x(\overline{X_1 \cdots X_n}) \rightsquigarrow X_0$ iff $X_0 \to x(\overline{X_1 \cdots X_n})$ is a rule of P (in pseudoterm form). We first prove that $X \overset{*}{\Rightarrow} \alpha$ in S iff $\rho(\alpha) \rightsquigarrow X$ in M, by induction on the depth of α.

(a) $\text{Depth}(\alpha) = 0 \Rightarrow \alpha = x \in A_0$, thus $X \to \alpha$ in S iff $X \Rightarrow x$ in S, since S is expansive, iff $X \to x$ is a rule in P iff $t_x \rightsquigarrow X$, by definition of M, iff $\rho(\alpha) \equiv \rho(x) \rightsquigarrow X$, by definition of ρ.

(b) Assume $\alpha = \langle U, V \rangle$ has pseudoterm representation $x(\alpha_1 \alpha_2 \cdots \alpha_n)$, $x \in A_n$, where $\{\alpha_i \mid 1 \leq i \leq n\}$ is the set of all sub-u-trees of $\alpha \ni (x, x_i) \in V$ and $x_i \in \alpha_i$ for $1 \leq i \leq n$. Our induction hypothesis is that if $\text{depth}(\alpha') < \text{depth}(\alpha)$ then $X \overset{*}{\Rightarrow} \alpha'$ in S iff $\rho(\alpha') \rightsquigarrow X$ in M. Note that we have $\text{depth}(\alpha_i) < \text{depth}(\alpha)$ for $1 \leq i \leq n$ by our definition of α. So, $X \overset{*}{\Rightarrow} \alpha = x(\overline{\alpha_1 \alpha_2 \cdots \alpha_n})$ in S iff $\exists\, X_1, \cdots, X_n \in B-A$ such that $X \Rightarrow x(\overline{X_1 \cdots X_n}) \overset{*}{\Rightarrow} x(\overline{\alpha_1 \alpha_2 \cdots \alpha_n})$ in S, since S is expansive, iff $X \to x(\overline{X_1 \cdots X_n})$ is a rule of P and $X_i \overset{*}{\Rightarrow} \alpha_i$ in S for $1 \leq i \leq n$ iff $t_x(X_1 \cdots X_n) \rightsquigarrow X$ by the definition of M and $\rho(\alpha_i) \rightsquigarrow X_i$, by the induction hypothesis, iff $\rho(\alpha) \rightsquigarrow X$, by the definition of ρ.

Now for $\alpha \in u_T A$, $\alpha \in L(M)$ iff $\rho(\alpha) \sim z$ in M iff $z \overset{*}{\Rightarrow} \alpha$ in S iff $\alpha \in L(S)$. Therefore $L(M) = L(S)$. Q.E.D.

Lemma 3.3 For every nondeterministic u-tree automaton one can effectively construct an equivalent deterministic u-tree automaton.

Proof Given nondeterministic UTA $M = \langle Q, t_1, \cdots, t_k, F \rangle$ for $Q = X_1, \cdots, X_j$ we construct deterministic $M' = \langle 2^Q, t_1', \cdots, t_k', F' \rangle$. We have $2^Q = X_1', \cdots X_{2^j}'$ where each X_i', $1 \leq i \leq 2^j$, denotes a separate subset of Q. We define t_i' by $t_i'(\overline{X_1' \cdots X_n'}) = \bigcup \{X_r \mid t_i(\overline{X_1 \cdots X_n}) \sim X_r$ for $X_1 \in X_1', \cdots, X_n \in X_n'\}$. We let $F' = \{X_i' \mid X_i' \cap F \neq \phi\}$. Clearly M' is deterministic since each t_i' is a function.

To complete the proof (i.e. to show that $L(M) = L(M')$) we first show, by induction on the depth of α, that $\rho_M(\alpha) \sim g$[5] iff $\rho_{M'}(\alpha) = G$ with $g \in G$.

(a) Assume $\text{depth}(\alpha) = 0 \Rightarrow \alpha = x \in A_0$. Now $\rho_M(\alpha) \sim g$ iff $t_x \sim g$ iff $t_x' = G$ with $g \in G$, by our definition of t_x', iff $\rho_{M'}(\alpha) = G$ with $g \in G$.

(b) Assume $\text{depth}(\alpha) > 0$ and $\alpha = \langle U, V \rangle$ with pseudoterm representation $x(\alpha_1 \alpha_2 \cdots \alpha_n)$, $x \in A_n$, where $\{\alpha_i \mid 1 \leq i \leq n\}$ is the set of all sub-u-trees of $\alpha \ni (x, x_i) \in V$ and $x_i \in \alpha_i$ for $1 \leq i \leq n$. Our induction hypothesis is that if $\text{depth}(\alpha') < \text{depth}(\alpha)$ then $\rho_M(\alpha) \sim g$ iff $\rho_{M'}(\alpha) = G$ with $g \in G$. Note that we have $\text{depth}(\alpha_i) < \text{depth}(\alpha)$ for $1 \leq i \leq n$ by our definition of α. Now $\rho_M(\alpha) \sim g$ iff $\exists\, X_1, \cdots, X_n \in Q \ni t_x(\overline{X_1 \cdots X_n}) \sim g$ and $\rho_M(\alpha_i) \sim X_i$, $1 \leq i \leq n$, by definition of ρ, iff $\rho_{M'}(\alpha_i) \sim X' \ni X_i \in X_i'$ for $1 \leq i \leq n$, our hypothesis, iff $t_x(\overline{X_1' \cdots X_n'}) = G$ with $g \in G$, by definition of t_i', iff $\rho_{M'}(\alpha) = G$ with $g \in G$.

We now observe that $\alpha \in L(M)$ iff $\rho_M(\alpha) \sim f \in F$ iff $\rho_{M'}(\alpha) = X'$ such that $f \in X'$ and $X' \in F'$, since $X' \cap F \neq \phi$, iff $\alpha \in L(M')$. Therefore $L(M) = L(M')$.

Q.E.D.

---

[5] We denote $\rho(\alpha)$ in machine M by $\rho_M(\alpha)$.

Theorem 3.1 For every regular u-tree grammar, G, one can effectively find a deterministic u-tree automaton, M, such that L(G)=L(M).

Proof This is a direct consequence of Lemmas 3.1, 3.2 and 3.3.

Theorem 3.2 Let $M=\langle Q,t_1,\cdots,t_k,F\rangle$ be a u-tree automaton over $\langle A,\sigma\rangle$. One can effectively construct a regular u-tree grammar $S=\langle B_S,\sigma_S,P_S,\Gamma_S\rangle$ over $\langle A,\sigma\rangle$ such that L(M)=L(S).

Proof Define tree automaton $M'=\langle Q',t_1',\cdots,t_k',F'\rangle$ by: $Q'=Q$, $F'=F$ and we include one rule of form $t_i'(X_1\cdots X_n)\sim X$ in M' iff $t_i(\overline{X_1\cdots X_n})\sim X$ in M (so we include one specific permutation from each permutation class). It is clear that for each u-tree $\alpha\epsilon L(M)$ one specific permutation of α will be accepted by M' and for each tree α' accepted by M' the u-tree $c_2^{-1}(\alpha')$ will be accepted by M. By Theorem 2.6 we know we can construct regular tree grammar G such that L(G)=L(M'). By Theorem 2.3 we know we can then construct a reduced, expansive regular tree grammar G' such that L(G')=L(G)=L(M'). Assume $G'=\langle B',\sigma',P',\Gamma'\rangle$ over $\langle A,\sigma\rangle$. G' will produce exactly one representation of each u-tree accepted by M. We can now define reduced, expansive RUTG $S=\langle B_S,\sigma_S,P_S,\Gamma_S\rangle$ over $\langle A,\sigma\rangle$ such that S will produce the u-trees in $c_2^{-1}(L(G'))$. We define $B_S=B'$, $\sigma_S=\sigma'$, and $\Gamma_S=\Gamma'$ so Γ_S contains a single one node axiom also. Since all productions in P' will be of the form either $X_0\to x$ or $X_0\to x(X_1\cdots X_n)$ for $x\epsilon A$ and $X_0,X_1,\cdots,X_n\epsilon B'-A$ we let $P=\{X_0\to x|X_0\to x\ P'\}\cup\{X_0\to x(\overline{X_1\cdots X_n})|X_0\to x(X_1\cdots X_n)\epsilon P'\}$. So we have L(S)=L(M). Q.E.D.

Theorem 3.3 The sets of u-trees generated by regular u-tree grammars are exactly the sets accepted by u-tree automata.

Proof Follows immediately from Theorems 3.1 and 3.2.

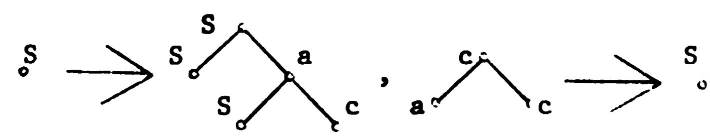
Theorem 3.4 Given regular u-tree grammar $G=\langle B,\sigma,P,\Gamma\rangle$, one can effectively construct regular tree grammar $G'=\langle B',\sigma',P',\Gamma'\rangle$ such that

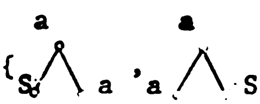
$L(G')=c_2(L(G))$, i.e. $t \in L(G')$ iff t is a representation of a u-tree in L(G).

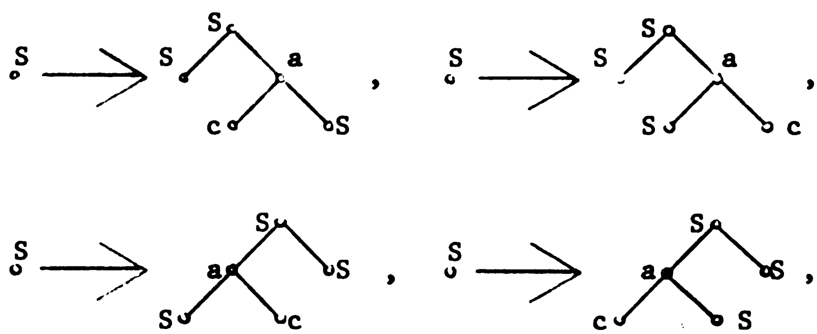
<u>Proof</u> Define $B'=B$, $\sigma'=\sigma$. A given u-tree will have only a finite number of representations, therefore since Γ is finite we can construct $\Gamma'=\{$representations of u-trees in $\Gamma\}$. WOLOG let the productions in P be $p_1, p_2, \cdots, p_n$ where $p_i=\phi_i \to \psi_i$ for $1 \le i \le n$. For each production $\phi_i \to \psi_i$ both ϕ_i and ψ_i will have a finite set of representations; say ϕ_i^r and ψ_i^r respectively. Now let $P=\bigcup_{i=1}^{n}(\phi_i^r \times \psi_i^r)$. Then the order of P, denoted $|P|$, will be $|P|=\sum_{i=1}^{n}|\phi_i^r|\cdot|\psi_i^r|$ which is finite. By our definition we know that G' will generate all representations of u-trees in G. Given $t \in L(G')$ we know that t is a representation of a u-tree in L(G) because no axioms or productions were included that were not legitimate representations.

Q.E.D.

<u>Example 3.4</u> Let $G=\langle A,\sigma,P,\{$  $\}\rangle$ be a regular u-tree grammar

with P = {  }

Then regular tree grammar $G'=\langle A,\sigma,P',\{$  $\}\rangle$ will generate all representations of u-trees in L(G) with

P'= { 

 }

<u>Corollary 3.1</u> Given regular u-tree grammar, G, one can effectively construct a deterministic tree automaton, T, such that $t \in L(T)$ iff t is a representation of a u-tree in L(G).

<u>Proof</u> Follows from Theorems 2.5 and 3.4.

CHAPTER 4

PATTERN RECOGNITION CONCEPTS

4.1 MOTIVATIONS

In Chapter 3 structures were developed whereby criterion (1) of an acceptable extension (see Section 3.1) to the theory of pattern recognition by tree automata is met. The purpose of this chapter is to develop the rest of the method so that it will become evident that criterion (2) is also met. Informally, our procedure when given a pattern will be to first represent its primitives as nodes of a d-tree (order will not matter), second find a string representation using bracketed notation and third present this string to a u-tree automaton for acceptance or rejection.

A secondary purpose of this chapter is to define some of the concepts and terms necessary for syntactic pattern recognition in general. There is, however, a major problem connected with such definitions. One must be careful not to "overdefine"; we want to keep the terminology broad enough so it can serve a wide variety of applications. With this difficulty in mind, the route adopted in this paper will be to only non-rigorously define some of the basic terms, these, essentially undefined terms, will be used to define the others. The definitions will be primarily non-mathematical. Section 4.2 is a discussion of the non-rigorously defined terms.

4.2 UNDEFINED TERMINOLOGY

The universe of discussion will indicate that physical area where a pattern of interest may occur. Typical universes might include: the subset from zero to infinity of a one-dimensional number line, a two-dimensional "frame" which can contain pictures, a finite two-dimensional grid of zeros and ones, quadrant 1 of an X/Y coordinate system, three-dimensional space etc.

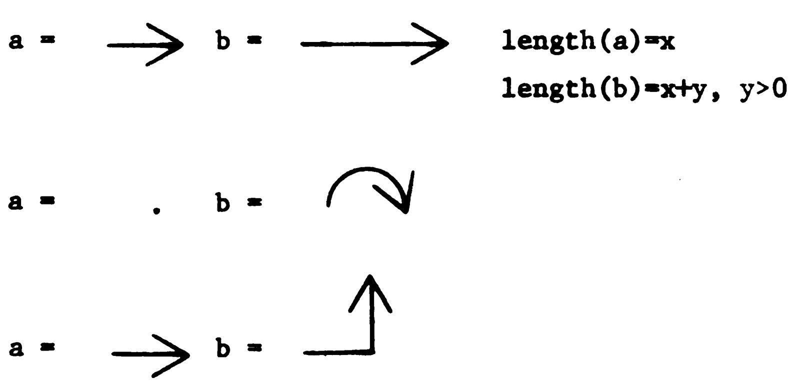
The term pattern will be used to designate some specific connected subset of the universe under discussion. The phrase pattern recognition will refer to classifying patterns as to whether or not they are included in a given pattern class (i.e. a given set of patterns).

Primitive types, themselves small patterns, will serve as the simple building blocks of which patterns may be composed. These might be points, directed line segments, three-dimensional cubes etc. We will say we can represent a pattern by a set of primitive types, P, if we can construct the pattern by repeated juxtaposition of members of P. The construction used to represent a pattern proceeds serially from a given start point; there may be restrictions as to what primitive types may precede and succeed others. A point of entry is the portion of a primitive type that must be placed adjacent to a serially preceding primitive type in a representation. A point of exit is that part that must be adjacent to a succeeding primitive type, if there is any. Points of exit or entry do not have to be single geometric points. Every primitive type except the start point must have a point of entry or it can never be applied but a point of exit is optional. Primitive types without a point of exit can have no successors.

A primitive is a single application of a primitive type while representing a figure. A primitive will be denoted by (x,y), where x indicates an application of primitive type x, and y serves as a unique identifier indicating which particular application.

Primitive type r will be said to be a structural part of primitive type s if r and s have the same points of entry and $r \subsetneq s$.

Example 4.1



In each of these cases "a" is a structural part of "b".

4.3 BUILDING U-TREES FROM PATTERNS

Definition 4.1 A primitive feature system is a triple $P_s = \langle S,P,D \rangle$ where:

(a) P is a finite set of primitive types.

(b) D is a finite set of informal descriptions of elements of P.

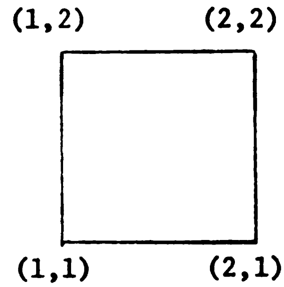
(c) $S \in P$, S is designated the start point.

(d) No primitive type (except S) is allowed to be a structural part of any other primitive type.

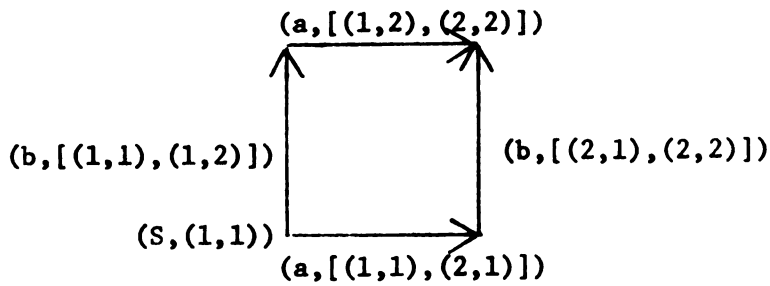
We normally refer to a primitive feature system simply as a primitive system.

Example 4.2 Given a pattern in quadrant 1 of an X/Y coordinate system. Let $P_{s0} = \langle S,P,D \rangle$, $P=\{S=\cdot, a=\longrightarrow, b=\uparrow\}$, D={(S is the closest point of the pattern to the origin),(length(a)=length(b)=w, but w may vary and will assume an appropriate size for a given figure),(a is horizontal), (b is vertical),(the point of entry for a and for b is the end away from the arrow point, the point of exit is the arrow point)}. In future examples where there should be no chance of misinterpretation we will usually only give a description for S. The rest of the descriptions will be clear from the geometric form of the primitive types. Using the previously defined P_{s0} let pattern F_1 be given:

F_1 =



Then the primitives will be:



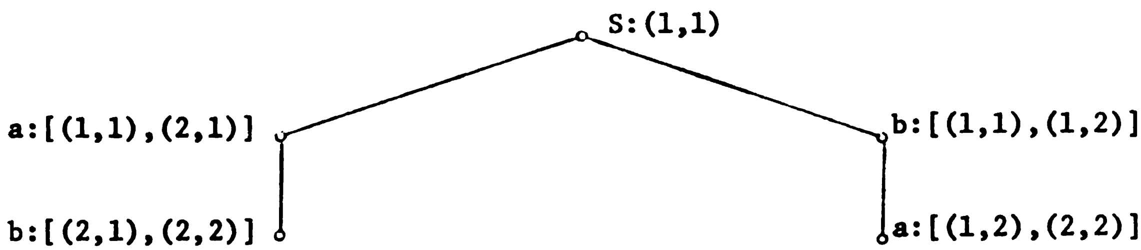
<u>Definition</u> <u>4.2</u> Given a primitive system $P_s = \langle S,P,D \rangle$, a corresponding d-tree, T to pattern, F, is a d-tree constructed using a representation of F by P_s with the following procedure:

(a) Root of T is S:identifier of S.

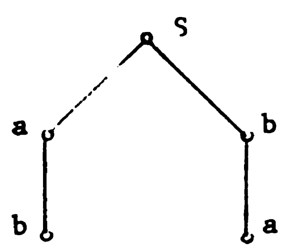
(b) Node $a_2{:}b_2$ is a successor of node $a_1{:}b_1$ in T iff the point of entry of primitive (a_2,b_2) coincides with the point of exit of primitive (a_1,b_1) in the representation.

Clearly, the corresponding d-trees when constructed in this manner will satisfy criterion (2) of our pattern recognition system. No reference needs to be made to anything, we just construct the d-tree taking all primitives as they occur in the representation. It is important to note that a given primitive, (a,x), may have its point of entrance coincide with more than one point of exit; in this case node a:x of a corresponding d-tree would occur on more than one root-to-frontier path.

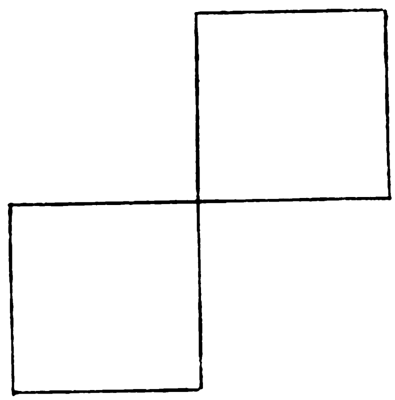
<u>Example</u> <u>4.3</u> Using P_{s0} and F_1 as defined earlier, the corresponding d-tree will be:



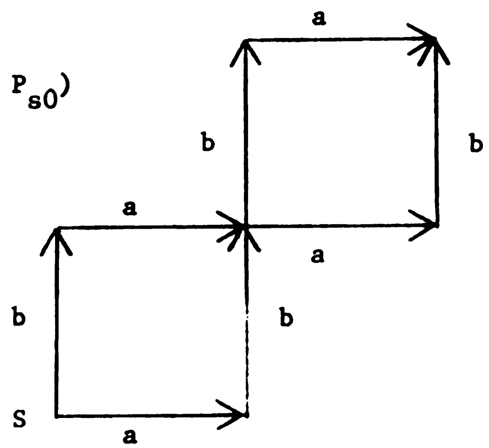
Where there is no possibility of misinterpretation we can leave off the unique identifiers and get:



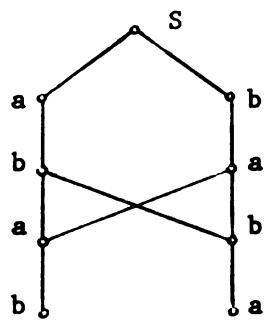
If we let F_2 =



then the primitives (using P_{s0})
will be:



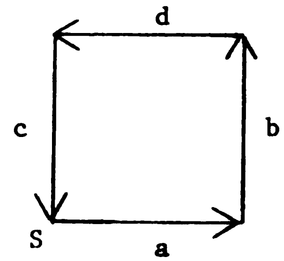
and the corresponding d-tree will be:



It should be clear exactly what primitive is represented by each node of this d-tree without the identifiers.

<u>Definition 4.3</u> A <u>circular</u> primitive system $P_s = \langle S,P,D\rangle$ is one in which for some pattern in the universe of discussion there is a corresponding d-tree containing a closed walk.

<u>Example 4.4</u> $P_{s4} = \langle S, \{S = \cdot, A = \longrightarrow, B = \uparrow, C = \downarrow, D = \longleftarrow\}, \{(S \text{ is the closest point to the origin in quadrant } 1)\}\rangle$. If F has primitive representation:



then S,(S,a),a,(a,b),b,(b,d),d,(d,c),c,(c,S),S is a closed walk in the corresponding d-tree so P_{s4} is circular.

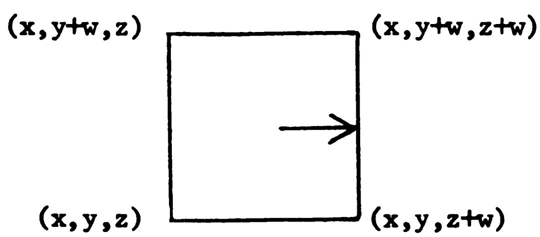
<u>Definition 4.4</u> A <u>noncircular</u> primitive system is one which is not circular. (i.e. No patterns can exist in the appropriate universe that can produce a corresponding d-tree containing a closed walk.)

<u>Example 4.5</u> The universe is the subset $[0,\infty)$ of one-dimensional space. $P_{s1} = \langle S, \{S = \cdot, a = \longrightarrow\}, \{(S \text{ is the point of the figure closest to } 0)\}\rangle$.
Then P_{s1} is noncircular.

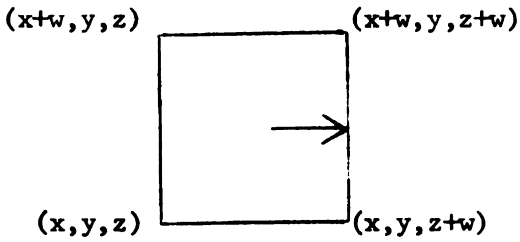
<u>Example 4.6</u> Universe = 3-dimensional space (in the area where $x,y,z \geq 0$).

$$P_{s2} = \langle S,P,D\rangle . \quad P = \{ S = \begin{array}{c}(x,y,z)\\ \Big| \\ (x,y,z+w)\end{array} \text{ or } \begin{array}{c}(x,y,z)\\ \Big| \\ (x,y+w,z)\end{array} \text{ or } \begin{array}{c}(x,y,z)\\ \Big| \\ (x+w,y,z)\end{array},$$

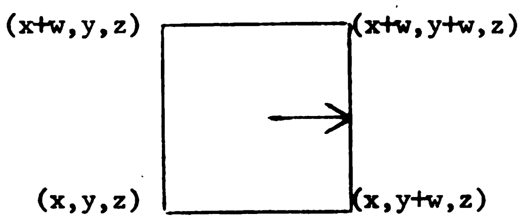
a =



b =



c =

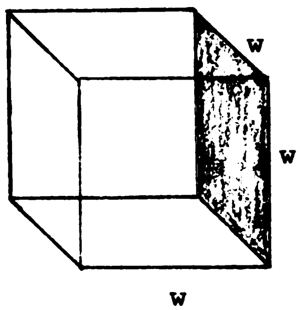


}

D={(S is the closest permissible line segment of the pattern to (0,0,0), i.e. the sum of the distances from the endpoints is the least),(a,b,c,S all share the same appropriate w),(the points of exit for a,b,c are the sides where the arrows point, points of entry are any other side)}. P_{s2}, then, is noncircular.

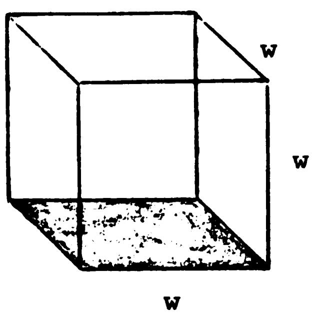
<u>Example</u> <u>4.7</u> $P_{s3}=\langle S,P,D\rangle$. S = a or b or c from Example 4.6.

P = { S, d =

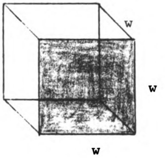


,

e =



,

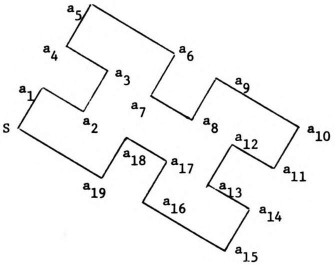
f =  }

D={(S is the closest, square, closed plane to the origin),(use the same appropriate w for S,d,e,f),(shaded sides of d,e,f are points of exit, any other sides are points of entry)}. Then P_{s3} is noncircular.

Definition 4.5 A primitive system is complete if it can represent all connected patterns in its domain. P_{s1} from Example 4.5 is complete.

Theorem 4.1 There is no noncircular complete primitive system in two-dimensional Euclidean space.

Proof Assume we have a complete primitive system, P_s. Then, in particular, P_s can represent pattern A, where A is placed with an appropriate orientation so S, as shown, will be the start symbol. (S must be a single point or P_s would not be able to represent a pattern which, itself, was a single point.)

A = 

Each symbol, a_i, $1 \leq i \leq 19$, represents an intersection of the line segments in A. Line segments $S - a_{19}$, $a_{16} - a_{15}$, $a_5 - a_6$ and $a_9 - a_{10}$ are of length 2c, all others are of length c. All angles are 90°. Since P_s is complete we know that it can represent not only A, but each line segment in A also. We will designate the list of primitive types that can represent the line segment $a_i - a_j$ by $\overline{a_i a_j}$. There are two cases to consider in the corresponding d-tree for A.

Case 1 $\exists$ primitive path $S \to S$. Then P_s is circular.

Case 2 $\nexists$ primitive path $S \to S$. This implies that there are at least two distinct primitive paths $S \to a_k$ for some $1 \leq k \leq 19$. Since the figure is symmetrical we can assume WOLOG that $k \leq 10$. What we need now, is to show that for any $k \leq 10$ there must be a list of primitives $S \to a_k \to S$. This will be a contradiction of assumed noncircularity. We will discuss the case for k=1 is some detail and then outline a similar argument for other values of k.

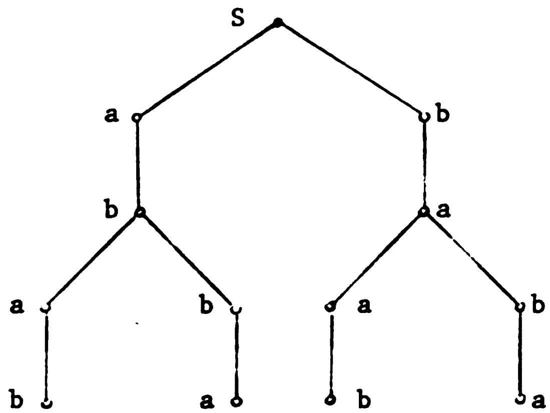
For k=1 there must be a primitive path leading directly from S to a_1 and another primitive path from S through a_{19}, then through a_{18} etc. leading to a_1. If we applied $\overline{a_{17}a_{16}}$ to a_1, however, we would have a path directly from S to a_1 and back to S $\otimes$ so for k=1 we get a contradiction. We now show that a contradiction arises similarly for all a_k with $k \leq 10$ and thus (by symmetry) for all a_k in the pattern.

If k=2	we	could	apply	$\overline{a_{17}a_{16}}$		then	$\overline{a_{14}a_{13}}$	therefore	$S \to a_2 \to S$ $\otimes$
" 3	"	"	"	$\overline{a_{17}a_{16}}$		"	$\overline{a_2 S}$	"	" a_3 " "
" 4	"	"	"	$\overline{a_{17}a_{16}}$		"	$\overline{a_1 S}$	"	" a_4 " "
" 5	"	"	"	$\overline{a_{17}a_{16}}$		"	$\overline{a_4 S}$	"	" a_5 " "
" 6	"	"	"	$\overline{a_{14}a_{13}}$	twice	"	$\overline{a_5 S}$	"	" a_6 " "

If k=7 we could apply $\overline{Sa_1}$ then $\overline{a_6S}$ therefore $S \to a_7 \to S$ ⊗
" 8 " " " $\overline{a_3a_4}$ " $\overline{a_7S}$ " " a_8 " "
" 9 " " " $\overline{a_{14}a_{13}}$ " $\overline{a_6S}$ " " a_9 " "
" 10 " " " $\overline{a_{14}a_{13}}$ twice " $\overline{a_9S}$ " " a_{10} " "

Therefore there cannot be a noncircular P_s. Q.E.D.

<u>Definition</u> <u>4.6</u> Given pattern, f, with corresponding d-tree, t; we say that u-tree, u, is a <u>corresponding</u> <u>u-tree</u> to f if u is the naturally corresponding u-tree to t. In Example 4.3 a corresponding u-tree for pattern F_2 will be:



Practically speaking we may observe that the corresponding u-tree is a theoretical construction only; it allows us to construct an accepting u-tree automaton, M, which will accept corresponding u-trees (thus accepting patterns) iff they are in L(M). We will never need to actually construct a u-tree when attempting to recognize a pattern: we can find the pseudoterm representation directly from the d-tree.

<u>Definition</u> <u>4.7</u> A set of patterns, T, is <u>definable</u> if there is a regular u-tree grammar, G, and a primitive system, P_s, such that given pattern t with y the corresponding u-tree for t, then $t \in T$ iff $y \in L(G)$.

Definition 4.7 does not mean that L(G) cannot contain u-trees which do not correspond to a figure in definable set T, in fact L(G) will often

contain such u-trees. These u-trees, however, cannot correspond to any physically possible pattern. See Example 4.10 for a case where this occurs.

Note that every finite set of patterns $\{F_1, F_2, \cdots, F_n\}$ will be definable since we could first define the u-tree productions, figure by figure, as $S \circ\!\!\rightarrow \begin{array}{c} S \\ P_i \end{array}$, one level trees, then define the primitive system to have the figures themselves, and S, as primitive types. (F_i corresponds to P_i.) This misses the spirit of what we mean by primitive type but it does not violate our definitions.

4.4 THE PATTERN RECOGNITION PROCEDURE

Theorem 4.2 Given definable set, T, there is a deterministic u-tree automaton, M, such that for pattern t with y the corresponding u-tree for t, then $t \in T$ iff $y \in L(M)$.

Proof Follows immediately from Theorem 3.1.

At this point we can describe our procedure for determining if a given pattern is in a given definable pattern set using an appropriate primitive system and u-tree grammar.

Pattern Recognition Procedure

Step(1) Given a pattern find the primitives which taken together constitute it.

Step(2) Build a d-tree with labels taken from the set of primitives.

Step(3) Present the pseudoterm corresponding to the d-tree to a previously constructed UTA for acceptance or rejection.

It seems appropriate to ask what parts of this procedure are completely defined and thus algorithms. Given a definable set of patterns, step (2) under most conditions (see Theorem 4.4 and related text) will be an algorithm. With a given UTA, M, step (3) is an algorithm defined by M. Step (1), however, in many cases is not completely defined. It is not the purpose of this thesis to describe methods of imposing primitive types, so we will just comment that probably a mathematical template matching involving "closest fit" could be used, but certainly a variety of methods might prove best for different applications. Of course, once a computer program has been written to find the primitives, the program itself will define a procedure.

Definition 4.8 Primitive type c has unique size means that any primitive (c,x) in a representation of a pattern must use the identical c with no variation in any of its measurements.

Lemma 4.1 Given u-trees $T_1=\langle U_1,V_1\rangle$, $T_2=\langle U_2,V_2\rangle$ with $\mathrm{root}(T_1)=\mathrm{root}(T_2)=r$ then T_1 not a sub-u-tree of $T_2 \Rightarrow \exists\, \mathrm{node}(a)$[1] $\in U_1\cap U_2$ such that $\{(a,x)\mid(a,x)\in V_1\}\neq\{(a,x)\mid(a,x)\in V_2\}$.

Proof Given the conditions of the lemma assume to the contrary that for all $a\in U_1\cap U_2, \{(a,x)\mid(a,x)\in V_1\}=\{(a,x)\mid(a,x)\in V_2\}$. In a u-tree every node can be reached on a path from the root. We first establish the claim that (given the assumption) $y\in U_1$ iff $y\in U_2$. This is shown by induction on the length of path $r,\cdots,y$.

(a) We know $r\in U_1\cap U_2$, and by the assumption $\{(r,x)\mid(r,x)\in V_1\}=\{(r,x)\mid(r,x)\in V_2\}$ for all $x\in U_i$ such that x can be reached from r by a path of length 1 thus $x\in U_{3-i}$ also.

(b) Assume claim true for all paths of length less than n. Now for $y\in U_i$ reached on a path of length n we will have path $r,(r,x_1),x_1,\cdots,x_{n-1},(x_{n-1},x_n),x_n=y$. Path $r,(r,x_1),x_1,\cdots,x_{n-2},(x_{n-2},x_{n-1}),x_{n-1}$ then is a

---

[1]We refer to a node by naming its label.

path of length n-1 in T_i, therefore, by our induction hypothesis, $x_{n-1} \in U_{3-i}$ also. Since (x_{n-1}, y) is in $V_i \Rightarrow$, by our assumption, that (x_{n-1}, y) is in $V_{3-i} \Rightarrow y \in U_{3-i}$.

So the claim is established and we have $U_1 = U_2$ therefore $U_1 \subseteq U_2$. Furthermore for $p_1 \in U_1$ and $(p_1, p_2) \in V_2$ we must have $(p_1, p_2) \in V_1$. Therefore $p_2 \in U_1$ and criterion (a) for definition of sub-u-tree (see definitions 3.9 and 3.10) is satisfied. $U_1 = U_2 \Rightarrow$ for all $a \in U_1$, $a \in U_1 \cap U_2 \Rightarrow$ for $a \in U_1$ and $x \in U_1$ $(a,x) \in V_1$ iff $(a,x) \in V_2$, by our assumption, $\Rightarrow$ criterion (b) for definition of sub-u-tree is satisfied $\Rightarrow T_1$ is a sub-u-tree of T_2 $\otimes$, so the assumption is false and the lemma is proved. Q.E.D.

<u>Theorem</u> <u>4.3</u> Given primitive system $P_s = \langle S,P,D \rangle$ with noncircular P and with the restriction that each $p \in P$ has unique size, then the corresponding d-tree to a given pattern, f, will be unique (to the extent that any two d-trees representing f will have identical naturally corresponding u-trees).

<u>Proof</u> Assume d-trees d_1 and d_2 representing f have corresponding u-trees T_1 and T_2 respectively with $T_1 \neq T_2$. This implies that (WOLOG say) T_1 is not a sub-u-tree of T_2. Assume $T_1 = \langle U_1, V_1 \rangle$ and $T_2 = \langle U_2 V_2 \rangle$. T_1 and T_2 must have the same root, S, this implies $\exists$ node$(a) \in U_1 \cap U_2$ such that $\{(a,x) \mid (a,x) \in V_1\} \neq \{(a,x) \mid (a,x) \in V_2\}$, by Lemma 4.1, but we can see that since size(a) is unique it must have the same point of exit (in relation to f) in both the representation for T_1 and that for T_2. Furthermore because of noncircularity this point can never have been reached previously during d-tree construction. Therefore every (a,x) in V_i (meaning that point of entry for x coincides with point of exit for a) must also be in V_{3-i} so that $\{(a,x) \mid (a,x) \in V_1\} = \{(a,x) \mid (a,x) \in V_2\}$ $\otimes$

contradicting the lemma. So the assumption is false and $T_1 = T_2$. Q.E.D.

There are other possible restrictions one might want to impose, rather than the unique size restriction of the theorem, in order that a

ue d-trees. In gener-

to represent patterns

priate restriction can

not true for circular

ge class of sets of

systems. The ob-

constructing d-trees

que d-trees. This is

stem let $P_{s6} = \langle S,P,D \rangle$.

— }

ngth of x = c),

This is to certify that the

thesis entitled

Unordered Tree Automata: Machines That Can Classify Patterns

presented by

Kenneth Leroy Williams

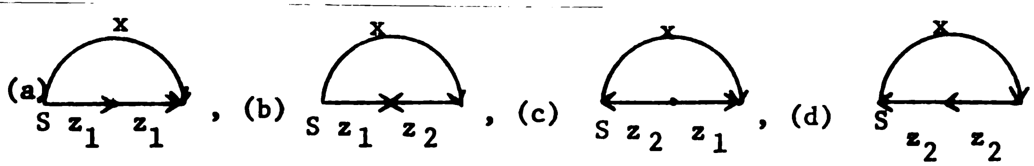
has been accepted towards fulfillment of the requirements for

Ph. D. degree in Computer Science

Carl V. Page
Major professor

Date August 7, 1973

O-7639



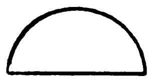
Constructing the d-trees on a breadth first basis will give either contradicting the lemma. So the assumption is false and $T_1=T_2$. Q.E.D.

There are other possible restrictions one might want to impose, rather than the unique size restriction of the theorem, in order that a given noncircular primitive system would give unique d-trees. In general if a noncircular primitive system will suffice to represent patterns of interest for a given application, then an appropriate restriction can be found to insure uniqueness. This, however, is not true for circular primitive systems and unfortunately there is a large class of sets of patterns that cannot be represented by noncircular systems. The obvious method, that of applying primitive types and constructing d-trees on a breadth first basis, will not always give unique d-trees. This is illustrated by the following example.

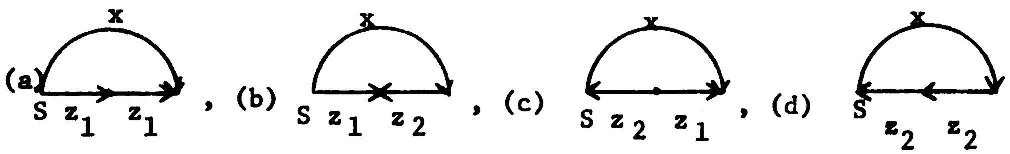
Example 4.8 In quadrant 1 of an X/Y coordinate system let $P_{s6}= \langle S,P,D \rangle$.

P = {S= . , x= , $z_1= \longrightarrow$, $z_2= \longleftarrow$ }

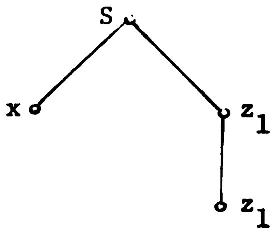
D={(S is the point closest to the origin), (cord length of x = c), (length(z_1)=length(z_2)=1/2 c)}.

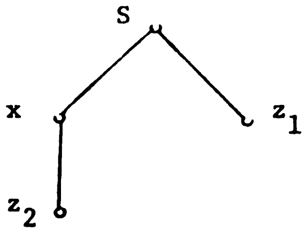
Let F = 

Imposing primitives gives either

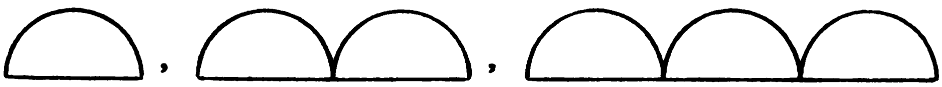


Constructing the d-trees on a breadth first basis will give either

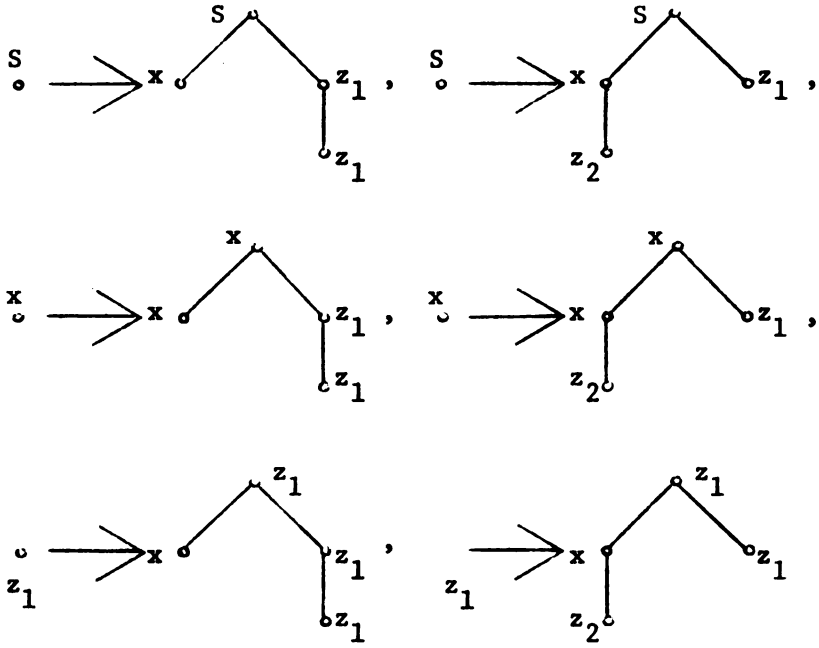


from (a) or


from (b) so there is no unique d-tree. We can however, in this case, define a regular u-tree grammar $G = \langle B,\sigma,P,\Gamma \rangle$ which will generate exactly those u-trees (with d-trees generated on a breadth-first basis) corresponding to the patterns in the set:

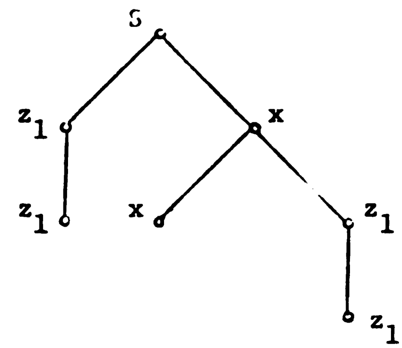
F = {

, ... etc.}

The grammar will have $B=\{S,x,z_1,z_2\}$, $\Gamma=\{S\cdot\}$ and

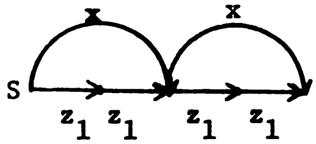
P = {


}

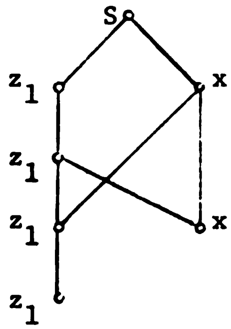
This grammar will generate many u-trees which are not corresponding u-trees for patterns in F, for example:

t = 

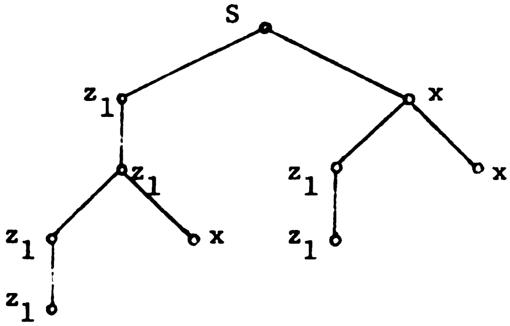
but every u-tree of this type will represent a physically impossible figure so there will be no harm done by our tree acceptor accepting d-trees we can't get anyway. If we try to draw a pattern corresponding to tree t we get:



Reversing ourselves and generating a corresponding d-tree (breadth-first) for this pattern, however, we get



which has corresponding u-tree



and since this is not the same u-tree as t we can see that t was not the corresponding u-tree for any real pattern. t must violate our definition of how the corresponding d-tree should be formed with every possible primitive branch.

Under certain conditions we can get unique d-trees from circular primitive systems if we are willing to do some exterior checking while building them. In the following theorem we assume one condition which is considered to be understood.

Unique Application Condition

We assume that there is a procedure for applying a primitive type so that if a given location of the pattern serves as the point of entry for the resulting primitive then the same primitive type will always be applied in the same way at the same location, so that exactly the same part of the pattern will be represented by it and the same location (in relation to the pattern) will be the point of exit. This seems a reasonable assumption since presumably we would implement this sort of scheme on a digital computer where the same machine instructions would always be executed in the same manner.

Theorem 4.4 Given a primitive system (possibly circular), under the Unique Application Condition, one can always construct a unique d-tree for a given pattern if a total ordering $(<,\simeq,>)$ can be defined on the unique primitive identifiers with $a\simeq b$ iff $a=b$.

Proof Given primitive system $P_s = \langle S,P,D \rangle$ we will prove the theorem by developing an algorithm which builds the d-tree as it imposes primitive types to give primitives. At each step there will be precisely one primitive which may be expanded (i.e. its successors found) thus the algorithm works by first taking an enumeration of the elements of P, say $p_1, p_2, \cdots, p_n$. (P must be finite, by definition.) In general p_i will be expanded before p_j if $i<j$. The problem is for $i=j$. Since each application of a primitive type has a unique identifier and since we have a total ordering on the identifiers we can make the decision to expand primitive (p_i, x_1) before (p_i, x_2) iff $x_1<x_2$ for $x_1, x_2 \in$ {unique identifiers}. The identifiers are to be assigned in such a manner that $x_1<x_2$ iff x_1 was applied not later than x_2. Thus the algorithm is partially breadth first since it tends to expand older nodes first.

For convenience the algorithm uses four sets, G, H, F and E (although only one is really necessary) as it builds representative d-tree, D. G will always contain the set of unexpanded primitives (or equivalently we may think of them as nodes). H, F and E will be used only for temporary storage.

Algorithm 4.1 Algorithm to generate unique d-tree, $D=\langle U,V\rangle$, (given above conditions) on a partially breadth first basis.

(1) Apply primitive type $S=p_i$, giving primitive (p_i, x_i). Root of $D \leftarrow p_i:x_1$. $G \leftarrow \{p_i:x_1\}$. $U \leftarrow \{p_i:x_1\}$.

(2) $H \leftarrow \{p_i:x_j \mid p_i:x_j \in G \text{ with } i=\min\{k \mid p_k:x_r \in G\}\}$. (H must be nonempty.)

(3) $F \leftarrow \{p_i:x_k \mid p_i:x_j \in H \Rightarrow x_k \leq x_j\}$. (F must contain exactly one element, say f, because of the total ordering.) $G \leftarrow G-F$.

(4) $E \leftarrow \{p_k:x_j \mid p_k:x_j$ is a result of expanding at f and each $x_j > x_i$ for all previously used $x_i\}$. $U \leftarrow U \cup E$. $V \leftarrow V \cup \{(f,w) \mid w\ E\}$. $G \leftarrow G \cup E$. If $G=\phi$ then exit. Else go to step (2).

The algorithm clearly must generate a unique d-tree. Q.E.D.

Definition 4.9 We say p is a projection from u-tree $u_1 \in u_T A_1$ to u-tree $u_2 \in u_T A_2$ if digraph u_2 is formed by relabeling the nodes of digraph u_1 with some relabeling function $p': A_1 \to A_2$. For sets of u-trees, U_1 and U_2, we say U_2 is a projection of U_1 if U_2 contains exactly the set of projections of u-trees in U_1.

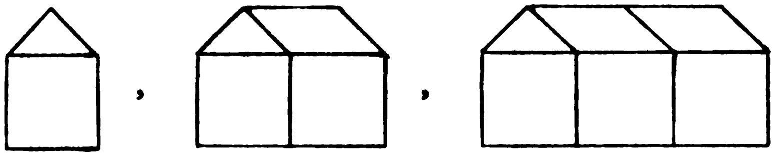
Theorem 4.5 Pattern set F definable implies that $\exists P_s, \Pi$ and G with Π a projection of the set of derivation trees generated by context-free grammar G, (we denote the set of projected derivation trees as $\Pi d(G)$), and P_s is a primitive system such that given pattern f, with y the corresponding u-tree for f, then $f \in T$ iff $y \in \Pi d(G)$.

Proof F definable $\Rightarrow \exists$ regular u-tree grammar G' such that L(G') satisfies the conditions for $\Pi d(G)$ in the theorem statement, by the definition of definable, $\Rightarrow \exists$ deterministic tree automaton M with L(M)= L(G) so L(M) satisfies the requirements of $\Pi d(G)$ in the theorem statement, by Corollary 3.1, $\Rightarrow \exists$ context-free grammar G with $\Pi d(G)$=L(M) so $\Pi d(G)$ satisfies the conditions of the theorem, by Theorem 2.1. Q.E.D.

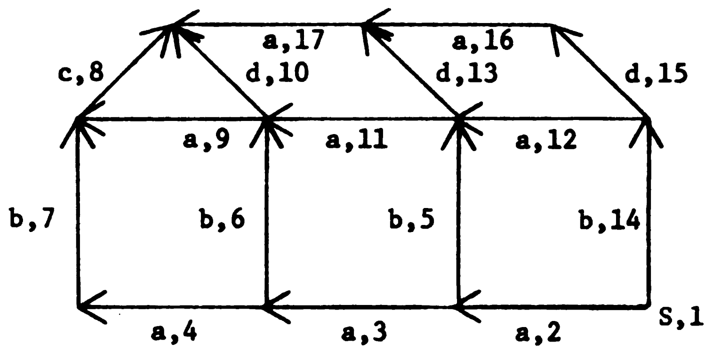
4.5 SOME EXAMPLES

Many infinite sets of patterns where each pattern is of finite size will be definable. In these cases we usually find that the patterns can each be broken down into connected subpatterns where the subpatterns are each definable and where any number of such connections may be made, giving larger and larger, but still finite patterns in the set. Recall Example 4.8. Such cases illustrate the advantages of the recursive manner in which regular u-tree grammars can generate u-trees.

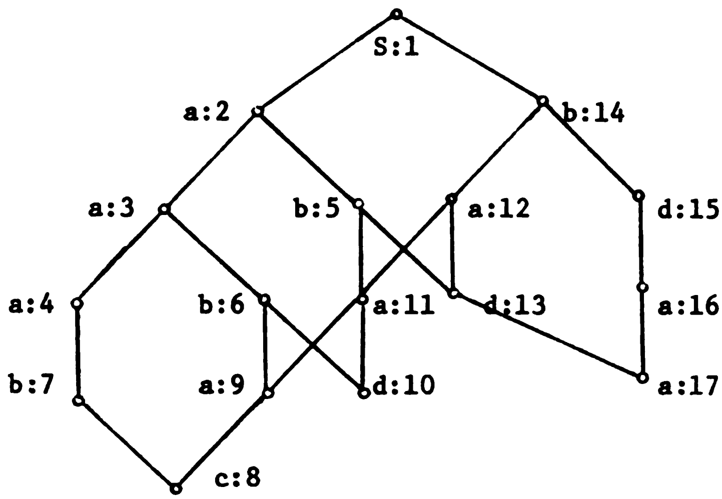
Example 4.9[2] It is desired to be able to recognize the set of houses,

T = {  etc. }

We can see that each house may be represented using only the primitive types: a= ←, b= ↑, c= ↗, d= ↖, S(the start point)=·, with the convention that S will represent the point in the lower right hand corner of a pattern. A specific house in the set may be represented as:



The numbers associated with each primitive serve to uniquely identify each specific application of a primitive type. We now construct the representative d-tree for the pattern:



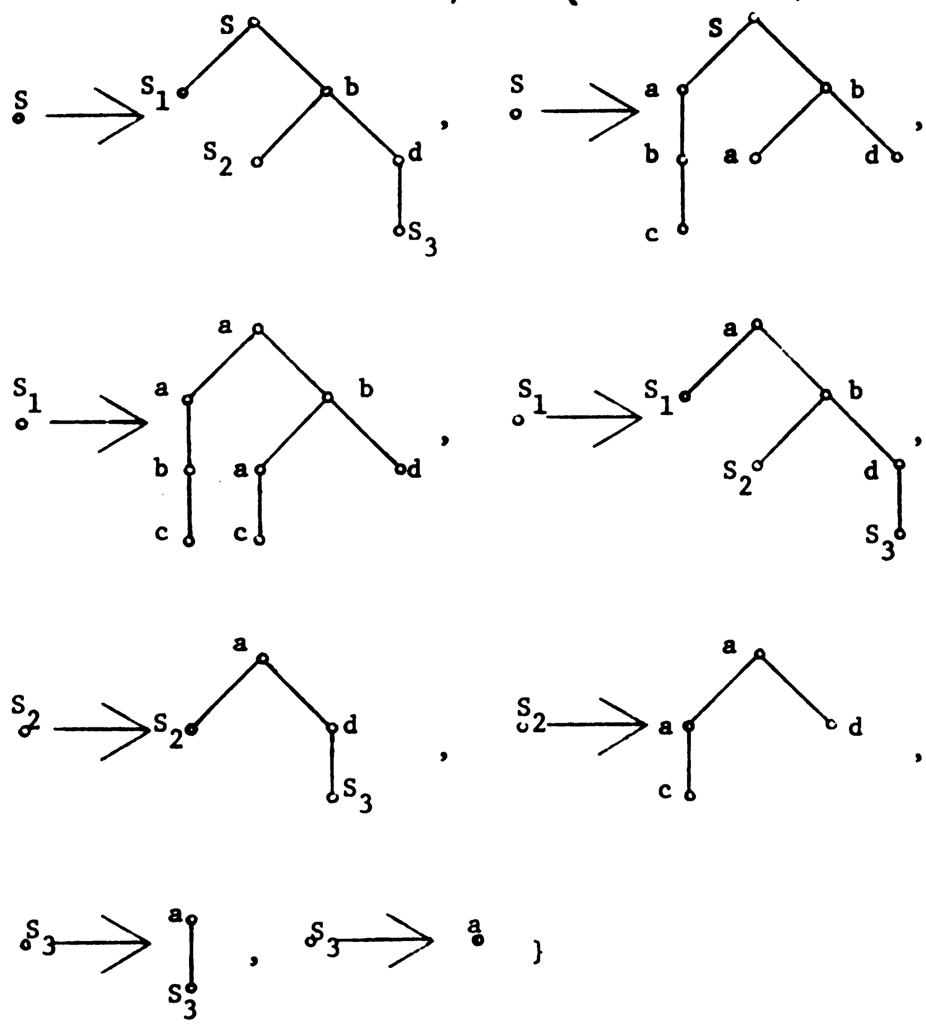
---

[2]This problem was also used as an example in Fu and Bhargava(20). A similar but more limited problem was first given by Shaw(38) and also appeared in Fu and Swain(18).

The d-tree is made by using each primitive label as the label for a node of the d-tree. We place arc $(a{:}x_i, b{:}x_j)$ in the d-tree if primitive $b{:}x_j$ can be directly reached from primitive $a{:}x_j$, always traveling with the flow of the arrows. Now that the d-tree is constructed we just present the corresponding term S(a(a(a(b(c))b(a(c)d))b(a(a(c)d)d(a)))b(a(a(a(c)-d)d(a))d(a(a))) to our previously constructed UTA for acceptance. To see that a UTA can be constructed that will accept terms only for d-trees corresponding to elements of the pattern set we observe that the following grammar will generate the corresponding u-trees. We can then use the procedure as indicated by Theorem 3.1 to construct the UTA.

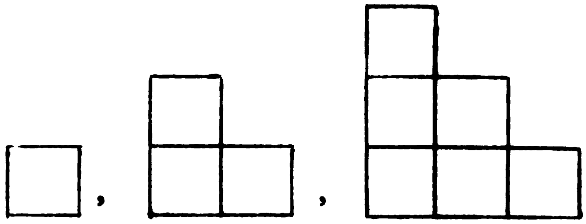
Let $G = \left\langle \{a,b,c,d,S,S_1,S_2,S_3\},\sigma,P,\{S\cdot\} \right\rangle$ over $\left\langle \{a,b,c,d,S\},\sigma' \right\rangle$ with

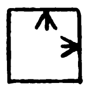
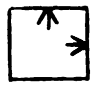
P = {

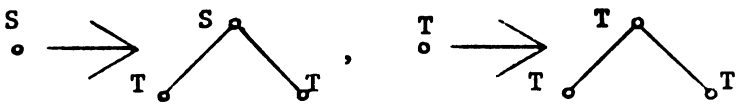


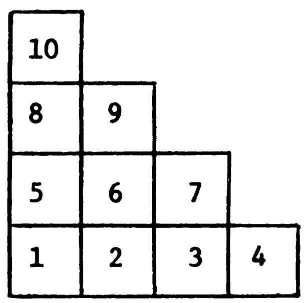
Another example using very different primitive types will be instructive.

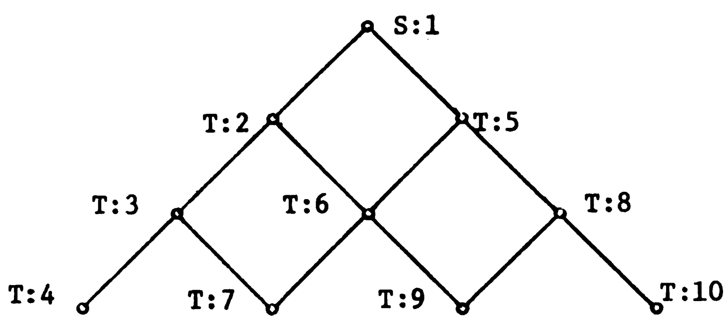
Example 4.10 Let the pattern set, T, be all isocoles right triangles with vertical and horizontal sides, constructed of grid blocks.

T = {  etc. }

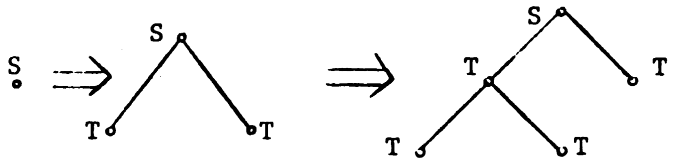
The primitive types are: S=  , T=  . S is the closest block to the lower left hand corner of a pattern. Points of exit for S and for T are the sides where arrows point, points of entrance are the other sides. Define the u-tree grammar to be G= $\langle \{S,T\},\sigma,P,\Gamma \rangle$

where P = {  }

The d-tree representing 

will be 

G can generate many u-trees that do not represent patterns in T, but these will not be the corresponding u-trees for physically possible patterns anyway, so no harm is done. Consider the derivation:



which produces a u-tree that does not correspond to an element of T. This u-tree, however, does not correspond to any possible pattern. Any attempt to construct such a pattern, while realizing that all possible connections must be shown in the u-tree illustrates this fact.

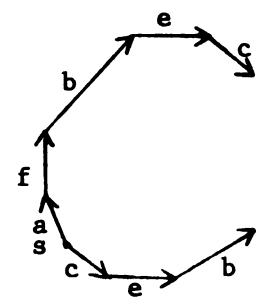
As mentioned earlier, one of the best features of this automatic method for pattern recognition is in the way it can recursively accept pseudoterms corresponding to recursively defined pattern sets.[3] As a last example we outline a possible use involving nonrecursive patterns. This example also illustrates the capability of the method for pattern classification as well as recognition. This method is probably not the most practical for printed characters; a vast amount of research has been done on this problem.

<u>Example 4.11</u> We want to recognize printed English capital letters. In order to keep the example small we will only work with three letters, C, G and O, and we will assume that these are written in an idea form for analysis. We will use primitive types: s=$\cdot$, a= $\nwarrow$, b= $\nearrow$, c= $\searrow$, d= $\swarrow$, e= $\longrightarrow$, f= $\uparrow$, g= $\longleftarrow$, h= $\downarrow$ with start point s taken as the closest point to the origin in quadrant 1 of an

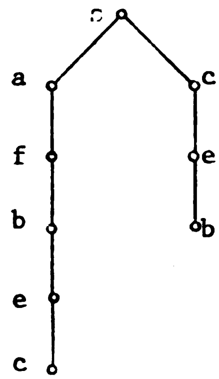
---

[3] In fact this recursive capability is one of the main advantages of any syntactic system.

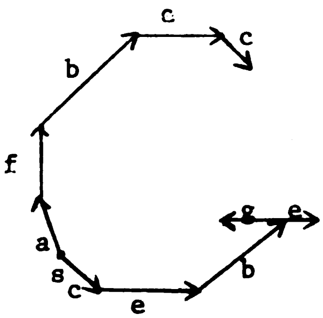
X/Y coordinate system. Since we have a clearly circular primitive system we will use the procedure outlined in Theorem 4.4 to ensure unique d-trees. The enumeration of primitive types will be as given above, so "a" is expanded before "b" which is expanded before "c" etc. Given ideal character C we would then represent it with this primitive system as:

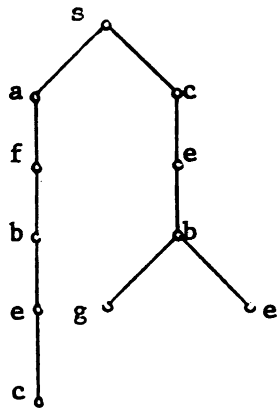


which has d-tree

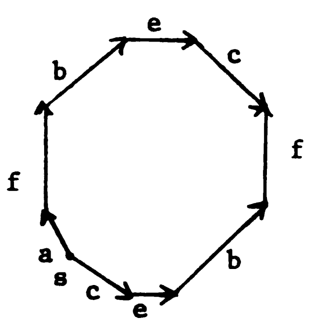


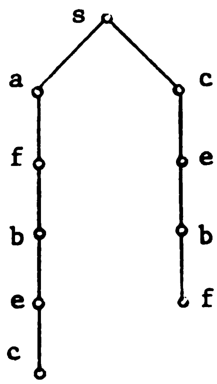
Representing G:



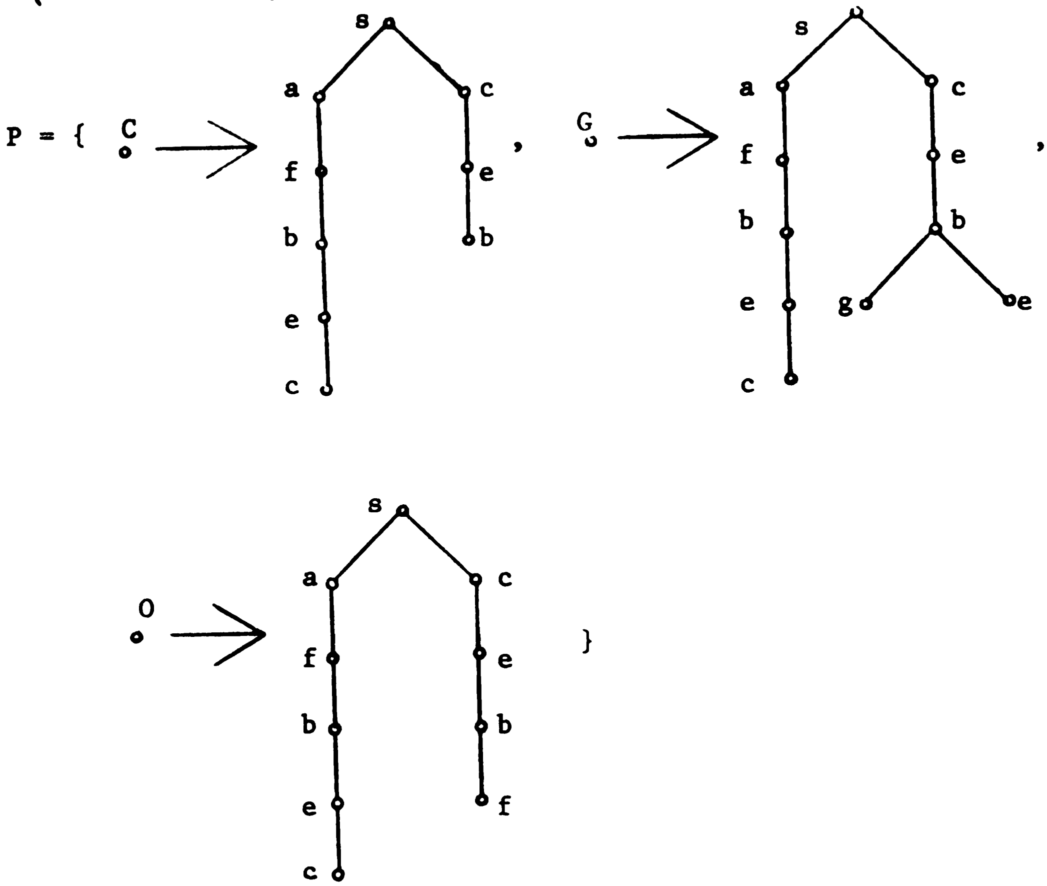


Representing O:

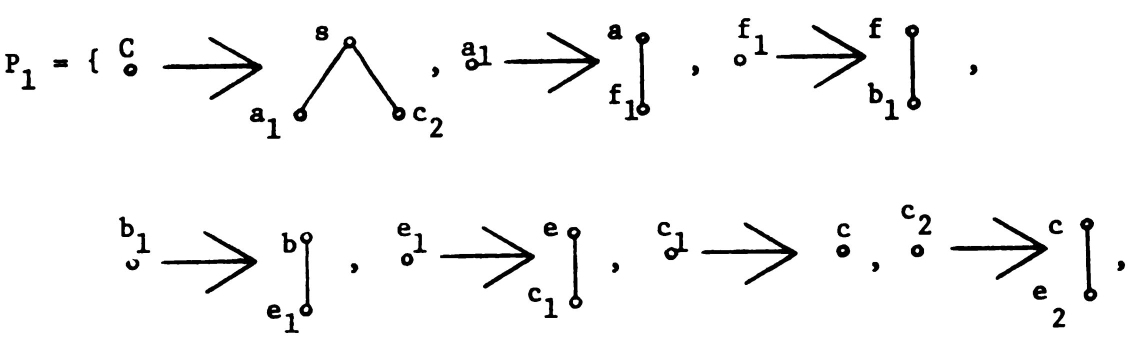


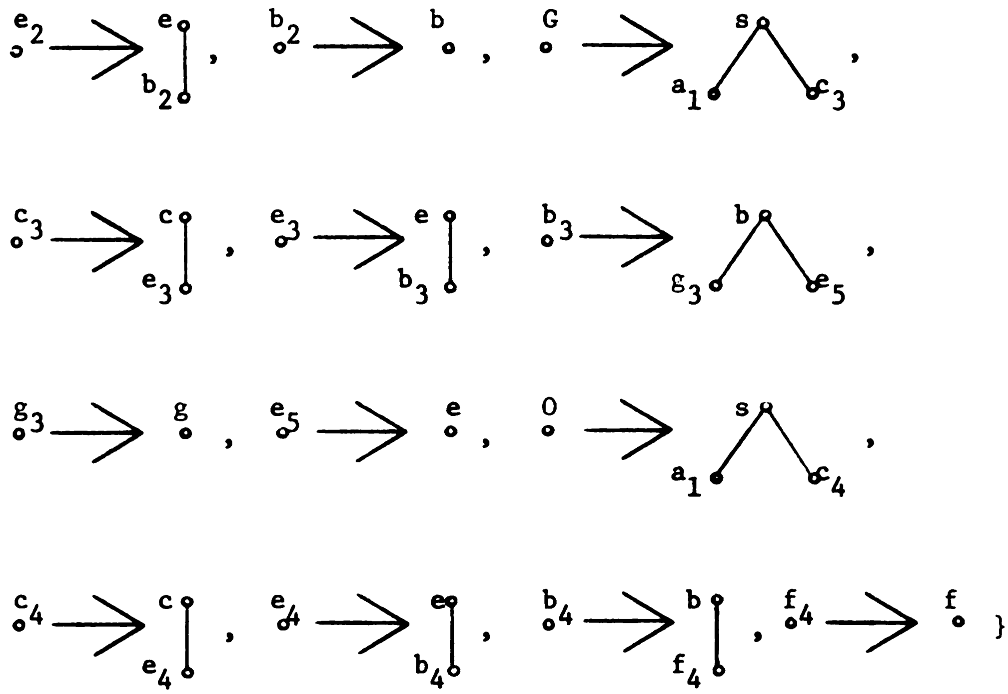


We define our u-tree grammar over $\left\langle \Sigma=\{s,a,b,c,d,e,f,g,h\},\sigma \right\rangle$ as $G=\left\langle B,\sigma',P,\{C,G,O\} \right\rangle$ with $B=\{C,B,O\} \cup \Sigma$ and



In order to define an accepting u-tree automaton we first need to find an equivalent expansive u-tree grammar over $\langle \Sigma,\sigma \rangle$. To get this we work through each production in P, introducing new nonterminals where necessary, to make expansive productions, giving $G=\left\langle B_1,\sigma',P_1,\{C\cdot,G\cdot,O\cdot\} \right\rangle$ with $B_1=\{a_1,b_1,b_2,b_3,b_4,c_1,c_2,c_3,c_4,e_1,e_2,e_3,e_4,e_5,f_1,f_4,g_3\} \cup B$ and





The deterministic u-tree automaton we construct will end in state C if the input pattern is C, state G if G and state O if O, so it will classify the input. It will not end in any one of these final states for any other input. Using the construction process outlined following Theorem 2.5 (modified for u-trees) we define $M=\left\langle B_1-\Sigma, t_s, t_a, t_b, t_c, t_d, t_e, t_f, t_g, t_h, \{C,G,O\}\right\rangle$. The t functions will be defined by:

$$t_s(\overline{a_1c_2})=C$$
$$t_s(\overline{a_1c_3})=G$$
$$t_s(\overline{a_1c_4})=O$$
$$t_a(f_1)=a_1$$
$$t_b(e_1)=b_1$$
$$t_b(\lambda)=b_2$$
$$t_b(\overline{g_3e_5})=b_3$$
$$t_b(f_4)=b_4$$
$$t_c(\lambda)=c_1$$

$$t_c(e_2)=c_2$$

$$t_c(e_3)=c_3$$

$$t_c(e_4)=c_4$$

$$t_e(c_1)=e_1$$

$$t_e(b_2)=e_2$$

$$t_e(b_3)=e_3$$

$$t_e(b_3)=e_4$$

$$t_e(\lambda)=e_5$$

$$t_f(b_1)=f_1$$

$$t_f(\lambda)=f_4$$

$$t_g(\lambda)=g_3$$

To see how M functions we will use Definition 3.16 to trace through an acceptance of pseudoterm s(a(f(b(e(c)))c(e(b(ge))) which represents the corresponding d-tree to an ideal input pattern, G.

$$\rho(s(a(f(b(e(c)))c(e(b(ge)))))=$$

$$t_s(\rho(a(f(b(e(c))))\rho(c(e(b(ge)))))=$$

$$t_s(t_a(\rho(f(b(e(c)))))t_c(\rho(e(b(ge)))))=$$

$$t_s(t_z(t_f(\rho(b(e(c)))))t_c(t_e(\rho(b(ge)))))=$$

$$t_s(t_a(t_f(t_b(\rho(e(c)))))t_c(t_e(t_b(\rho(g)\rho(e)))))=$$

$$t_s(t_a(t_f(t_b(t_e(\rho(c)))))t_c(t_e(t_b(t_g(\lambda)t_e(\lambda)))))=$$

$$t_s(t_a(t_f(t_b(t_e(t_c(\lambda)))))t_c(t_e(t_b(g_3e_5))))=$$

$$t_s(t_a(t_f(t_b(t_e(c_1))))t_c(t_e(b_3)))=$$

$$t_s(t_a(t_f(t_b(e_1)))t_c(e_3))=$$

$$t_s(t_a(t_f(b_1))c_3)=$$

$$t_s(t_a(f_1)c_3)=$$

$$t_s(a_1c_3)=$$

$$G$$

So M does end in final state G and the input is recognized as G.

CHAPTER 5

PROPERTIES OF U-TREE GRAMMARS AND LANGUAGES

In this chapter it is shown that a simplification of the automatic machines used to accept u-trees is possible. It is also shown that the string languages consisting of the frontiers of all representations of the u-trees in a u-tree language form an interesting class of languages.

5.1 SIMPLIFICATION OF ACCEPTING MACHINES

Although u-tree automata have appealing theoretical properties when used as indicated in Chapters 3 and 4, they have several obvious drawbacks as practical machines. Difficulties occur whether the machine is to be actually constructed of hardware or simply simulated on a digital computer. The machines are complex. It would be at best difficult to build hardware or software with state/move instructions which could map equivalence classes of strings of states into individual states. The direct method of simulating this procedure is to scan a given string of states comparing each state with the various equivalence classes to determine which class the string belongs to. (One might, of course, find ways of encoding states which would lead to more efficient methods.)

There is another implementation consideration of these machines that is not immediately evident from their definition. Some mechanism must be provided to monitor the levels of recursion as a machine accepts strings. At each step a record must be maintained containing all preceding information that could relate to future state conversions. Certainly a pushdown stack or simulation of a pushdown stack would be the appropriate device for this. When we consider the complexity of the u-tree automata definition as well as the above implementation considerations it is clear that they are very unwieldy machines to work with.

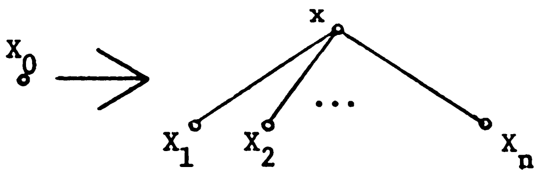
Since the languages accepted by u-tree automata are really languages of strings (the pseudoterms that represent u-trees) it seems worthwhile to investigate other types of machines to accept the same languages, machines whose definition may not arise so naturally from the definition of regular u-tree grammars, but which will be easier to implement.

We begin the search for accepting machines with an examination of nondeterministic pushdown automata (abbreviated PDA) and finish by considering deterministic PDA.

Theorem 5.1 For any regular tree grammar, G, one can construct a nondeterministic pushdown automaton that will accept exactly those pseudoterms representing elements of L(G).[1]

Proof Given G over $\langle A,\sigma\rangle$ one can construct reduced, expansive regular tree grammar $G'=\langle B,\sigma',P,\{Z\cdot\}\rangle$ over $\langle A,\sigma\rangle$ with $L(G')=L(G)$, by Theorem 2.3. One can then construct phrase structure grammar $G''=\langle V_N=B-A, V_T=A\cup\{),(\},P',\{Z\}\rangle$ such that P' will have $X_0\to x(X_1\cdots X_n)$ for each

[1]An equivalent result was proved in Brainerd(7).



in P. L(G") will then be exactly the set of pseudoterms representing elements of L(G')=L(G). Each of the productions in P' will have a single nonterminal in B-A as its LHS (since G' is expansive) thus G" is a context-free grammar, therefore one can construct a nondeterministic PDA which will accept exactly L(G") which is exactly those pseudoterms representing elements of L(G).

Q.E.D.

<u>Corollary 5.1</u> Given any regular u-tree grammar, M, one can construct a nondeterministic PDA that will accept exactly those pseudoterms representing elements of L(M).

<u>Proof</u> Given M, consider regular tree grammar G that simulates M, from Theorem 3.4. Then apply Theorem 5.1 to G. Q.E.D.

<u>Corollary 5.2</u> Given any regular u-tree automaton, M, one can construct a nondeterministic PDA that will accept exactly L(M).

<u>Proof</u> Follows from Theorem 3.2 and Corollary 5.1.

So we can always construct a nondeterministic PDA to take the place of a u-tree automaton. Nondeterministic PDAs, however, are themselves difficult to implement; there might be cases where it would be easier to make a deterministic u-tree automaton than a nondeterministic PDA. We would like to be able to use a deterministic PDA. This would involve an easy implementation or simulation.

<u>Theorem 5.2</u> Given a context-free phrase structure grammar, G, there is a deterministic PDA which will accept a string, t, iff t is a pseudoterm representation of a parse tree generated by G.

<u>Proof</u> Let $G=\langle V_N,V_T,P,\gamma_0\rangle$, with $P=\{(p_1)\colon A_{j_1}\to w_1$

$$(p_2)\colon A_{j_2}\to w_2$$

$$\vdots$$

$$(p_n)\colon A_{j_n}\to w_n\}$$

with $|V_N|=m$,[2] $\Sigma=V_N\cup V_T$, $A_{j_i}\in V_N$ for $1\leq i\leq n$.

Let $G'=\langle V_N',V_T',P',\alpha_0\rangle$ where $V_N'=\{\alpha_i|1\leq i\leq m,\ \alpha_i\notin\Sigma\}$ so the nonterminals V_N' will be a set of new symbols.

Let $V_T'=\Sigma\cup\{),(\}$.[3] Each production p_i' in P' will be obtained from the corresponding production p_i in P by the following procedure:

(1) Replace each occurrence of any nonterminal $\gamma_j\in p_i$ by the corresponding new nonterminal α_j, giving new production x_i.

(2) For each production $x_i\colon\alpha_j\to w'$, let p_i' be $\alpha_j\to\gamma_j(w')$.

Now grammar G' will generate exactly the set of pseudoterm representations of parse trees generated by G. P' will have no equal RHSs since every RHS is composed of both the LHS and the RHS (transformed by step 1) of a production in P and the same production will not occur twice in P. Furthermore when parsing a string in G' it is always clear exactly when a substitution should be made; it will be immediately after reaching symbol ")" as input is scanned from left to right. The sequence to be substituted for will be exactly $\gamma_k(w')$ for $\gamma_k\in V_N$, $w'\in(V_T\cup V_N')^*$. So G' is LR(0) hence a deterministic PDA can be constructed to accept exactly L(G')[4] and L(G') is exactly the pseudoterms representing parse trees generated by G. Q.E.D.

---

[2] $|V_N|$ indicates the number of elements in V_N.

[3] We assume $),(\notin\Sigma$.

[4] See Hopcroft and Ullman(22).

Example 5.1 Given context-free grammar $G=\langle V_N=\{S,B\},V_T=\{0,1,x\},P,S\rangle$ with

$P = \{S \rightarrow 0S0$

$S \rightarrow 1B1$

$B \rightarrow x\}$

Let $G'=\langle V_N'=\{\alpha_S,\alpha_B\},V_T'=\{S,B,0,1,x,")","("\},P',\alpha_S\rangle$. To generate P' we will have after step (1):

$\alpha_S \rightarrow 0\alpha_S 0$

$\alpha_S \rightarrow 1\alpha_B 1$

$\alpha_B \rightarrow x$

After step (2) we get:

$P' = \{\alpha_S \rightarrow S(0\alpha_S 0)$

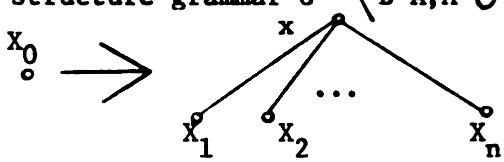
$\alpha_S \rightarrow S(1\alpha_B 1)$

$\alpha_B \rightarrow B(x)\}$

and G' will generate the pseudoterms representing derivation trees produced by G.

Theorem 5.3 Given regular tree grammar, G, in expansive form with no equal right hand sides in its tree productions, there is a deterministic PDA which will accept exactly the set of pseudoterms representing elements of L(G).

Proof Given $G=\langle B,\sigma',P,\{Z\cdot\}\rangle$ over $\langle A,\sigma\rangle$ we can construct context-free phrase structure grammar $G'=\langle B-A,A'\cup\{),(\},P',\{Z\}\rangle$ such that for each



in P, P' will contain $X_0 \rightarrow x(X_1 \cdots X_n)$. We can then make the same observations about parsing G' as those made for the G' in Theorem 5.2. Q.E.D.

Corollary 5.3 Given regular u-tree grammar, G, in expansive form with no equal right hand sides in its u-tree productions, there is a deterministic PDA which will accept exactly the set of pseudoterms representing elements of G.

Proof Given G consider regular tree grammar G' that simulates G (see Theorem 3.4). G' will still be reduced, expansive and contain no equal RHSs. Then apply Theorem 5.3 to G'. Q.E.D.

We now state the strongest result of this section.

Theorem 5.4 Given any regular tree grammar, G, one can construct a deterministic PDA that will accept exactly those pseudoterms representing elements of L(G).

Proof We will outline the construction for the deterministic PDA, M', so that it will be clear that M' will accept set L, the correct set of pseudoterms.[5] We assume G is defined over ranked alphabet $\langle A,\sigma\rangle$. Let $M=\langle B,t_1,t_2,\cdots,t_m,S\rangle$ be a deterministic tree automaton which accepts L by final state. The existence of machine M is established by Theorem 2.5. We define $M'=\langle K,\Sigma=A\cup\{(),(\},\Gamma=\Sigma\cup B,\delta,k_0,\#,F\rangle$ with $F\subset k$ and F containing a matching element k_s for each $s\in S$ so $F=\{k_s|s\in S\}$. M' will end in state k_s and accept an input iff M would end in state s and accept the same input (there is a single exception where M' will end in a failure state, k_{fail}, if M would have failed anyway). It is, of course, understood that if M' has no applicible move function defined for a (state,input,stack) triple then it halts and the present state is the final state. Note that the pushdown stack will show symbol # for empty stack.

---

[5]Here we use the Hopcroft and Ullman(22) (continued on next page -)

---

[5](continued) formalization for deterministic pushdown automata, namely a deterministic PDA M is a system $\langle K,\Sigma,\Gamma,\delta,q_0,\#,F\rangle$ where

(1) K is a finite set of states.

(2) Σ is the finite input alphabet.

(3) Γ is the finite pushdown alphabet.

(4) $q_0 \in K$ is the initial state.

(5) $\# \in \Gamma$ is the start symbol which initially appears on the pushdown store.

(6) $F \subseteq K$ is the set of final states.

(7) δ is a mapping from $K\times(\Sigma\cup\{\lambda\})\times\Gamma$ (λ is null input) to finite subsets of $K\times\Gamma^*$ such that

(a) $\delta(q,a,z)$ contains at most one element.

(b) $\delta(q,\lambda,z)$ contains at most one element.

(c) If $\delta(q,\lambda,z)$ is not empty, then $\delta(q,a,z)$ is empty for all $a\in\Sigma$.

If $\delta(q,a,z)=(p,\gamma)$ we adopt the convention that the rightmost symbol of γ will be placed highest on the store and the leftmost symbol lowest on the store.

We say M accepts by final state if an input is accepted iff M halts in a final state.

In order to define δ we first divide the various t function/input pairs defined by M into distinct sets.

$T_0=\{t_j(\lambda)\,|\,t_j(\lambda)\in M\}$

$T_1=\{t_j(w)\,|\,t_j(w)\in M,\ |w|=1\}$

$\vdots$

$T_r=\{t_j(w)\,|\,t_j(w)\in M,\ |w|=r$, r is the maximum length of any w such that $t_j(w)\in M\}$

Note that each of $T_0,T_1,\cdots,T_r$ will be finite.

(In the following move function definitions any input or stack string containing ")" or "(" will be indicated with quotation marks around it. Input on stack strings not containing these symbols will not be enclosed in quotation marks.)

Step 1 M' will begin reading an input (left to right) from its input tape and will immediately push the input onto the stack, so we define

$\delta(k_0,x,\#)=(k_0,\#x)$ for all $x\in A$.

Whenever it reads an "(" it pushes it down and enters state $k_($ so we define

$\delta(k_0,$"(",$x)=(k_($,"x(") for all $x\in A$.

We then define

$\delta(k_(,y,$"(")$=(k_1,$"(y") for each $y\in A$.[6]

Note that neither "(" or ")" is a valid input symbol to follow "(" for an element of L. We also define

$\delta(k_i,y,x)=(k_{i+1},xy)$ for each $y\in A$, for each $x\in B\cup A$, for $1\le i\le r-1$,

[6]M' as defined here will not adequately handle the special case where there are acceptable single-node trees. It is easy to change Step 1 slightly if these also are to be accepted.

$\delta(k_i, "(", x) = (k_(, "x(")$ for each $x \in B \cup A$, for $1 \leq i \leq r-1$,

$\delta(k_i, ")", x) = (k_{),i}, x)$ for each $x \in B \cup A$, for $1 \leq i \leq r$.

The total effect of δ as defined on states $k_), k_0, k_1, \cdots, k_r, k_{),1}, k_{),2}, \cdots, k_{),r}$ will be to read the initial input, left to right, until the first ")" is read. The input is pushed directly onto the stack and M' will end in state $k_{),j}$ for "(w)" being the last j+2 input symbols with $|w|=j$ and $), (\notin w$.

<u>Step 2</u> In state $k_{),j}$ we will use the pairs in T_0 to determine the next states as M' pops the stack down to the last "(". M' will now simulate M as M would apply $t_x(\lambda)$ for x on the frontier of a tree. M' keeps a record of what is replaced by what state it ends in. We define

$\delta(k_{),j}, \lambda, x) = (k_{),j,X_1}, \lambda)$ for all j such that $T_j \neq \phi$, for each $x \in B \cup A$ with $X_1 = t_x(\lambda)$ if $[t_x(\lambda)] \in T_0$

$X_1 = x$ otherwise.

$\delta(k_{),j,X_1}, \lambda, x) = (k_{),j,X_1,X_2}, \lambda)$ for all j such that $j \geq 2$ and $T_j \neq \phi$, for each $x \in B \cup A$ with $X_2 = t_x(\lambda)$ if $[t_x(\lambda)] \in T_0$

$X_2 = x$ otherwise.

$\vdots$

$\delta(k_{),j,X_1,\cdots,X_{j-1}}, \lambda, x) = (k_{),j,X_1,\cdots,X_j}, \lambda)$ for all j such that $T_j \neq \phi$, for all $X_i \in B$ with $1 \leq i \leq j-1$, for each $x \in B \cup A$ with

$X_j = t_x(\lambda)$ if $[t_x(\lambda)] \in T_0$

$X_j = x$ otherwise.

The δ functions taken where $[t_x(\lambda)] \notin T_0$ will simulate M as it makes no substitution in a string of states for a state which is itself a substitution.

Note that we are only defining a finite number of δ functions and states $k_{),j,\cdots}$ since $B \cup A$ is finite and we are only considering state encodements for input strings of length less than or equal to j for $1 \leq j \leq r$.

After M' applies these δ functions defined above, it will be in state $k_{),j,X_1,\cdots,X_j}$ and the stack (bottom to top) must contain (for a valid input string): "$\#w_1(w_2\cdots w_n($".

Step 3 We define

$\delta(k_{),j,X_1,\cdots,X_j},\lambda,"(")=(k_{(,j,X_1,\cdots,X_j},\lambda)$ for all $j\in\{i|i\geq 1,\ T_i\neq\phi\}$, for all strings $X_1\cdots X_j$ from B of length j,

in order to pop the top "(". We define

$\delta(k_{(,j,X_1,\cdots,X_j},\lambda,x)=(k_{Y,Y},\lambda)$ for $[t_x(X_jX_{j-1}\cdots X_1)]\in T_j$ with $t_x(X_jX_{j-1}\cdots X_1)=Y$ in M, for all $j\in\{i|i\geq 1,\ T_i\neq\phi\}$.

These are the functions (and they will be functions since M was deterministic) that actually determine what substitutions to make to simulate the functions in M, except those in T_0 (which are already made). So M' will end in state $k_{Y,Y}$ whenever a substitution from M, resulting in Y, could be made.

Step 4 It is now necessary to determine if the stack is empty. We define

$\delta(k_{Y,Y},\lambda,\#)=(k_Y,\#)$ for each $k_{Y,Y}$ defined in step 3 and

$\delta(k_{Y,Y},\lambda,x)=(k_{Y'},xY)$ for each $x\in\Gamma$, for each $k_{Y,Y}$ defined in step 3.

Step 5 M' ends in state k_Y or $k_{Y'}$ whenever a substitution (except those indicated by T_0) has been made, followed by a move function from step 4. If there is no more input on the input tape and state k_Y is reached (indicating the stack was empty) then k_Y is the final state reached and if $k_Y\in F$ the input is accepted. M' must check for more input in state k_Y: if there is any, M' can move directly to a fail state. We define

$\delta(k_Y, x, \#) = (k_{fail}, \#)$ for all $x \in \Sigma$.

If state $k_{Y'}$ is reached M' has made a substitution and must continue. If there is more input it is necessary to determine exactly how many symbols have been placed on the stack since the last "(", in order to know in what state $k_1, k_2, \cdots, k_r$ M' should now be in. We define the next group of states and move functions to accomplish this. M' also must know what the next input is.

At this point the top of the stack must contain an element of B, in fact the element Y when in state $k_{Y'}$. Define

(a) $\delta(k_{Y'}, x, Y) = (k_{x,Y}, \lambda)$ for each $x \in A$, for each k_Y as defined in Step 4

(b) $\delta(k_{Y'}, \text{"("}, Y) = (k_{(}, \text{"Y("})$ for each $k_{Y'}$ as defined in Step 4. (Note that $k_{(}$ and its associated move functions were defined in Step 1.)

(c) $\delta(k_{Y'}, \text{")"}, Y) = (k_{P,Y}, \lambda)$ for each $k_{Y'}$ as defined in Step 4.

<u>Step 6</u> We first define δ for case(c) from Step 5. Define

$\delta(k_{P,X_1}, \lambda, x_2) = (k_{P,X_1 x_2}, \lambda)$ for each $X_1 \in B$, for each $x_2 \in B \cup A$,

$\delta(k_{P,X_1,x_2}, \lambda, x_3) = (k_{P,X_1,x_2,x_3}, \lambda)$ for each $X_1 \in B$, for $x_2, x_3 \in B \cup A$,

$\vdots$

$\delta(k_{P,X_1,\cdots,x_{j-1}}, \lambda, x_j) = (k_{P,X_1,\cdots,x_j}, \lambda)$ for each $X_1 \in B$, for all $x_i \in B \cup A$ with $2 \le i \le j$ for $1 \le j \le r$,

$\delta(k_{P,X_1,\cdots,x_j}, \lambda, \text{"("}) = (k_{),j}, \text{"(}x_j \cdots x_2 X_1\text{"})$ for each $X_1 \in B$, for all $x_i \in B \cup A$ with $2 \le i \le j$ for all j such that $T_j \ne \phi$.

These δ functions will leave M' in state $k_{),j}$ with everything necessary replaced on the stack.

<u>Step 7</u> For case(a) in Step 5 we define

$\delta(k_{x_1,x_2},\lambda,x_3)=(k_{x_1,x_2,x_3},\lambda)$ for each $x_1,x_2,x_3\in B\cup A$,

$\vdots$

$\delta(k_{x_1,\cdots,x_{j-1}},\lambda,x_j)=(k_{x_1,\cdots,x_j},\lambda)$ for all $x_i\in B\cup A$ with $1\leq i\leq j$ for $1\leq j\leq r$,

$\delta(k_{x_1,\cdots,x_j},\lambda,"(")=(k_j,"(x_j\cdots x_1")$ for all j such that $T_j\neq\phi$, and for all $x_i\in B\cup A$ with $1\leq i\leq j$.

Move functions for state k_j were defined in Step 1 so in all three cases generated by Step 5 M' wil enter states whose move functions are previously defined. We have defined deterministic PDA M' so that it will clearly simulate deterministic tree automaton M and thus accept exactly those pseudoterms representing elements of L(G). Q.E.D.

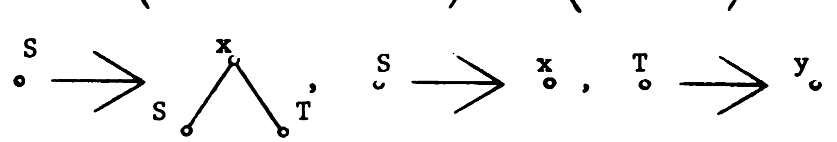
Corollaries 5.4 and 5.5 follow Theorem 5.4 in exactly the same manner that Corollaries 5.1 and 5.2 follow Theorem 5.1.

<u>Corollary 5.4</u> Given any regular u-tree grammar, G, one can construct a deterministic PDA that will accept exactly those pseudoterms representing elements of L(G).

<u>Corollary 5.5</u> Given any regular u-tree automaton, M, one can construct a deterministic PDA that will accept exactly L(M).

An example will illustrate the development of a deterministic PDA to accept L(G) for a regular tree grammar.

<u>Example 5.2</u> $G=\langle\{S,T,x,y\},\sigma',P,\{S\cdot\}\rangle$ over $\langle A=\{x,y\},\sigma\rangle$.

P = {


}

G is a reduced, expansive, regular tree grammar. An accepting tree automaton constructed as in the proof of Theorem 5.4 will be:

$M=\langle B=\{S,T\},t_x,t_y,\{S\}\rangle$ with: $t_x(ST)=S$, $t_x(\lambda)=S$, and $t_y(\lambda)=T$.

Our accepting deterministic PDA will be:

$M' = \left\langle K, \Sigma=\{x,y,")","("\}, \Gamma=\Sigma \cup B, \delta, k_0, \#, \{k_S\} \right\rangle$.

The T_i sets used to determine some of the functions in M' will be:

$$T_0=\{t_x(\lambda), t_y(\lambda)\}$$

$$T_2=\{t_x(ST)\}$$

The value of r will be 2.

We now define K and δ.

Generated by Step 1:

1. $\delta(k_0,x,\#)=(k_0,\#x)$ *
2. $\delta(k_0,y,\#)=(k_0,\#y)$
3. $\delta(k_0,"(",x)=(k_(,"x(")$ *
4. $\delta(k_0,"(",y)=(k_(,"y(")$
5. $\delta(k_(,x,"(")=(k_1,"x(")$ *
6. $\delta(k_(,y,"(")=(k_1,"y(")$
7. $\delta(k_1,x,x)=(k_2,xx)$
8. $\delta(k_1,x,y)=(k_2,yx)$
9. $\delta(k_1,x,S)=(k_2,Sx)$
10. $\delta(k_1,x,T)=(k_2,Tx)$
11. $\delta(k_1,y,x)=(k_2,xy)$ *
12. $\delta(k_1,y,y)=(k_2,yy)$
13. $\delta(k_1,y,S)=(k_2,Sy)$
14. $\delta(k_1,y,T)=(k_2,Ty)$
15. $\delta(k_1,"(",x)=(k_(,"x(")$ *
16. $\delta(k_1,"(",y)=(k_(,"y(")$
17. $\delta(k_1,"(",S)=(k_(,"S(")$
18. $\delta(k_1,"(",T)=(k_(,"T(")$

19. $\delta(k_1,\text{")"},x)=(k_{),1},x)$

20. $\delta(k_1,\text{")"},y)=(k_{),1},y)$

21. $\delta(k_1,\text{")"},S)=(k_{),1},S)$

22. $\delta(k_1,\text{")"},T)=(k_{),1},T)$

23. $\delta(k_2,\text{")"},x)=(k_{),2},x)$

24. $\delta(k_2,\text{")"},y)=(k_{),2},y)$ *

25. $\delta(k_2,\text{")"},S)=(k_{),2},S)$

26. $\delta(k_2,\text{")"},T)=(k_{),2},T)$

Generated by Step 2:

27. $\delta(k_{),1},\lambda,x)=(k_{),1,S},\lambda)$

28. $\delta(k_{),1},\lambda,y)=(k_{),1,T},\lambda)$

29. $\delta(k_{),1},\lambda,S)=(k_{),1,S},\lambda)$

30. $\delta(k_{),1},\lambda,T)=(k_{),1,T},\lambda)$

31. $\delta(k_{),2},\lambda,x)=(k_{),2,S},\lambda)$

32. $\delta(k_{),2},\lambda,y)=(k_{),2,T},\lambda)$ *

33. $\delta(k_{),2},\lambda,S)=(k_{),2,S},\lambda)$

34. $\delta(k_{),2},\lambda,T)=(k_{),2,T},\lambda)$

35. $\delta(k_{),2,S},\lambda,x)=(k_{),2,S,S},\lambda)$

36. $\delta(k_{),2,S},\lambda,y)=(k_{),2,S,T},\lambda)$

37. $\delta(k_{),2,S},\lambda,S)=(k_{),2,S,S},\lambda)$

38. $\delta(k_{),2,S},\lambda,T)=(k_{),2,S,T},\lambda)$

39. $\delta(k_{),2,T},\lambda,x)=(k_{),2,T,S},\lambda)$ *

40. $\delta(k_{),2,T},\lambda,y)=(k_{),2,T,T},\lambda)$

41. $\delta(k_{),2,T},\lambda,S)=(k_{),2,T,S},\lambda)$ *

42. $\delta(k_{),2,T},\lambda,T)=(k_{),2,T,T},\lambda)$

Generated by Step 3:

43. $\delta(k_{),2,S,S},\lambda,\text{"("})=(k_{(,2,S,S},\lambda)$

44. $\delta(k_{),2,T,S},\lambda,\text{"("})=(k_{(,2,T,S},\lambda)$ *

45. $\delta(k_{),2,S,T},\lambda,\text{"("})=(k_{(,2,S,T},\lambda)$

46. $\delta(k_{),2,T,T},\lambda,\text{"("})=(k_{(,2,T,T},\lambda)$

47. $\delta(k_{(,2,T,S},\lambda,x)=(k_{S,S},\lambda)$ *

Generated by Step 4:

48. $\delta(k_{S,S},\lambda,\#)=(k_S,\#)$ *

49. $\delta(k_{S,S},\lambda,S)=(k_{S'},SS)$

50. $\delta(k_{S,S},\lambda,T)=(k_{S'},TS)$

51. $\delta(k_{S,S},\lambda,x)=(k_{S'},xS)$

52. $\delta(k_{S,S},\lambda,y)=(k_{S'},yS)$

53. $\delta(k_{S,S},\lambda,\text{")"})=(k_{S'},\text{")S"})$

54. $\delta(k_{S,S},\lambda,\text{"("})=(k_{S'},\text{"(S"})$ *

Generated by Step 5:

55. $\delta(k_S,x,\#)=(k_{fail},\#)$!

56. $\delta(k_S,y,\#)=(k_{fail},\#)$!

57. $\delta(k_S,\text{"("},\#)=(k_{fail},\#)$!

58. $\delta(k_S,\text{")"},\#)=(k_{fail},\#)$!

59. $\delta(k_{S'},x,S)=(k_{x,S},\lambda)$

60. $\delta(k_{S'},y,S)=(k_{y,S},\lambda)$ *

61. $\delta(k_{S'},\text{"("},S)=(k_{(},\text{"(S"})$

62. $\delta(k_{S'},\text{")"},S)=(k_{P,S},\lambda)$

Generated by Step 6:

63. $\delta(k_{P,S},\lambda,x)=(k_{P,S,x},\lambda)$

64. $\delta(k_{P,S},\lambda,y)=(k_{P,S,y},\lambda)$

65. $\delta(k_{P,S},\lambda,S)=(k_{P,S,S},\lambda)$

66. $\delta(k_{P,S},\lambda,T)=(k_{P,S,T},\lambda)$

67. $\delta(k_{P,T},\lambda,x)=(k_{P,T,x},\lambda)$

68. $\delta(k_{P,T},\lambda,y)=(k_{P,T,y},\lambda)$

69. $\delta(k_{P,T},\lambda,S)=(k_{P,T,S},\lambda)$

70. $\delta(k_{P,T},\lambda,T)=(k_{P,T,T},\lambda)$

71. $\delta(k_{P,S,x},\lambda,\text{"("})=(k_{),2},\text{"(xS"})$

72. $\delta(k_{P,S,y},\lambda,\text{"("})=(k_{),2},\text{"(yS"})$

73. $\delta(k_{P,S,S},\lambda,\text{"("})=(k_{),2},\text{"(SS"})$

74. $\delta(k_{P,S,T},\lambda,\text{"("})=(k_{),2},\text{"(TS"})$

75. $\delta(k_{P,T,x},\lambda,\text{"("})=(k_{),2},\text{"(xT"})$

76. $\delta(k_{P,T,y},\lambda,\text{"("})=(k_{),2},\text{"(yT"})$

77. $\delta(k_{P,T,S},\lambda,\text{"("})=(k_{),2},\text{"(ST"})$

78. $\delta(k_{P,T,T},\lambda,\text{"("})=(k_{),2},\text{"(TT"})$

Generated by Step 7:

79. $\delta(k_{S,x},\lambda,\text{"("})=(k_2,\text{"(xS"})$

80. $\delta(k_{S,y},\lambda,\text{"("})=(k_2,\text{"(yS"})$

81. $\delta(k_{S,S},\lambda,\text{"("})=(k_2,\text{"(SS"})$

82. $\delta(k_{S,T},\lambda,\text{"("})=(k_2,\text{"(TS"})$

83. $\delta(k_{T,x},\lambda,\text{"("})=(k_2,\text{"(xT"})$

84. $\delta(k_{T,y},\lambda,\text{"("})=(k_2,\text{"(yT"})$

85. $\delta(k_{T,S},\lambda,\text{"("})=(k_2,\text{"(ST"})$

86. $\delta(k_{T,T},\lambda,\text{"("})=(k_2,\text{"(TT"})$

87. $\delta(k_{x,x},\lambda,\text{"("})=(k_2,\text{"(xx"})$

88. $\delta(k_{x,y},\lambda,\text{"("})=(k_2,\text{"(yx"})$

89. $\delta(k_{x,S},\lambda,\text{"("})=(k_2,\text{"(Sx"})$

90. $\delta(k_{x,T},\lambda,"(")=(k_2,"(Tx")$

91. $\delta(k_{y,x},\lambda,"(")=(k_2,"(xy")$

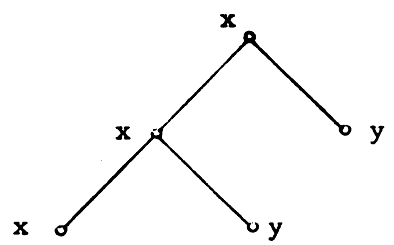
92. $\delta(k_{y,y},\lambda,"(")=(k_2,"(yy")$

93. $\delta(k_{y,S},\lambda,"(")=(k_2,"(Sy")$ *

94. $\delta(k_{y,T},\lambda,"(")=(k_2,"(Ty")$

These 94 move functions were generated by directly following the rules given in the construction; we will see that many of them are not necessary.

Using M' as developed we show the recognition procedure for x(x(xy)y) which is the pseudoterm representation for:



Stack contents are shown in order from bottom to top.

	Stack	Input	State
Initially	#	x(x(xy)y) ↑	k_0
Apply rule: 1	#x	(x(xy)y) ↑	k_0
3	#x(	x(xy)y) ↑	$k_($
5	#x(x	(xy)y) ↑	k_1
15	#x(x(	xy)y) ↑	$k_($
5	#x(x(x	y)y) ↑	k_1
11	#x(x(x(xy	)y) ↑	k_2
24	#x(x(xy	y) ↑	$k_{),2}$
32	#x(x(x	y) ↑	$k_{),2,T}$
39	#x(x(	y) ↑	k),2,T,S
44	#x(x	y) ↑	$k_{(,2,T,S}$

		Stack	Input	State
Apply rule:	47	#x(	y) ↑	$k_{S,S}$
	54	#x(S	y) ↑	$k_{S'}$
	60	#x(	) ↑	$k_{y,S}$
	93	#x(Sy	) ↑	k_2
	24	#x(Sy		$k_{),2}$
	32	#x(S		$k_{),2,T}$
	41	#x(		$k_{),2,T,S}$
	44	#x		$k_{(,2,T,S}$
	47	#		$k_{S,S}$
	48	#		k_S

Since there are no applicible rules for $(k_S,\lambda,\#)$ M' is finished. k_S is a final state so the input is accepted as it should be.

Input x(x(xy)y) is an exhaustive test of necessary move functions for this example; any move function not used here was not necessary (with the exceptions noted below). The ones used are indicated with an *. Only fifteen move functions were used so our deterministic PDA could be greatly simplified by not including any of the others except those which immediately take M' from an accepting final state on any input. These are indicated with an ! beside them. They must be maintained or the machine could end in an accepting final state with more input still to come, but with no move function defined. Even with these included, M' only requires 19 states.

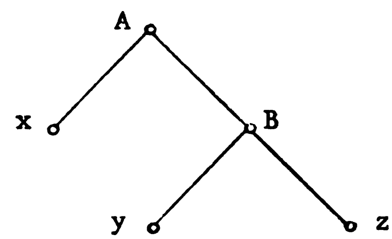
5.2 LANGUAGE PROPERTIES OF THE FRONTIERS OF U-TREE LANGUAGES

Definition 5.1 The frontier of a u-tree representation is simply the frontier of the tree used in the representation.

Definition 5.2 The frontier of a u-tree is the set of frontiers of representations for the u-tree.

Example 5.3

U-tree T =



The frontier of this u-tree representation is xyz.

The frontier of T is {xyz,yzx,xzy,zyx}.

Definition 5.3 The string language, denoted $L_s(G)$, of a regular u-tree grammar, G, is the union of all the frontiers of u-trees in L(G).

For every u-tree grammar we have defined a language of strings. We use the following notation as we investigate this class of languages:

Θ = the class of regular sets.

Δ = the class of context-free sets (i.e. the context-free languages).

Γ = the class of string languages generated by regular u-tree grammars.

Lemma 5.1 $\Theta \not\subseteq \Gamma$.

Proof Let B={01}. Then B is regular (every finite set of strings is regular). Assume $B \in \Gamma, \Rightarrow \exists$ a regular u-tree grammar, G, such that $L_s(G)=B$, $\Rightarrow$ we can apply a sequence of u-tree productions from G giving a u-tree representation, A, with string frontier 01,$\Rightarrow \exists$ some node, say a, in A with two successors, the left eventually leading to 0 and the right to 1. Since A is only a representation of a u-tree $\Rightarrow$

$\exists$ another representation, A', such that A' can also be generated by the same sequence of u-tree productions but a in A' will have its left and right successors reversed $\Rightarrow$ 10 is the string frontier of A' $\Rightarrow$ $10 \in L_s(G)$ $\otimes$ therefore $B \notin \Gamma$. Q.E.D.

<u>Lemma</u> <u>5.2</u> $\Delta \nsubseteq \Gamma$.

<u>Proof</u> $\Theta \nsubseteq \Gamma$ and $\Theta \subseteq \Delta \Rightarrow \Delta \nsubseteq \Gamma$.

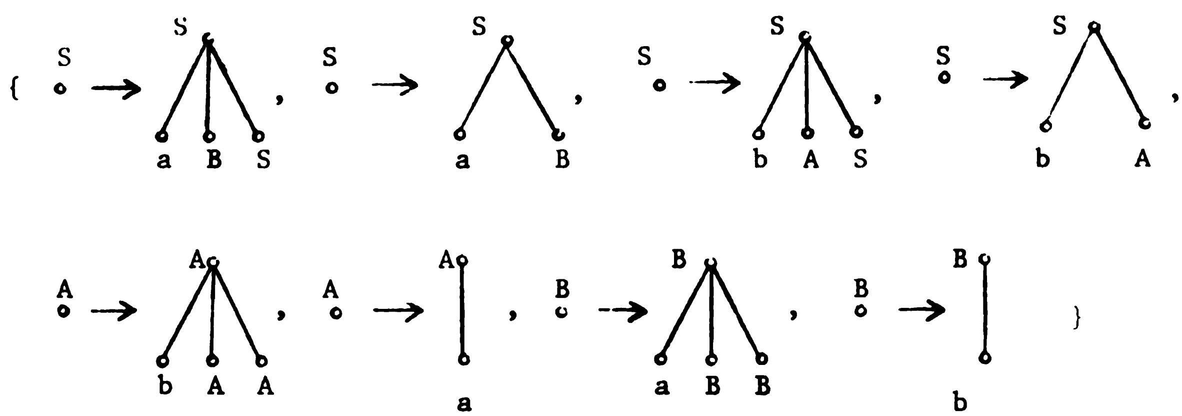
<u>Lemma</u> <u>5.3</u> $\Gamma \subset \Delta$.

<u>Proof</u> Let $A \in \Gamma \Rightarrow \exists$ a regular tree grammar, G, with frontier(G)=A, by Theorem 3.4, $\Rightarrow \exists$ a deterministic tree automaton, M, such that L(M)= L(G), by Theorem 2.5, $\Rightarrow$ A is a projection of a context-free language, by Theorem 2.1, $\Rightarrow A \in \Delta$ (shown in (22)) therefore $\Gamma \subseteq \Delta$. Lemma 5.2 shows that the inclusion must be proper. Q.E.D.

<u>Lemma</u> <u>5.4</u> $\Gamma \nsubseteq \Theta$.

<u>Proof</u>

Claim(1) Regular tree grammar $G_1 = \langle \{S,A,B,a,b\}, \sigma_1', P_1, \{S\cdot\} \rangle$ over $\{S,A,B,a,b\}$ with P_1 =



will produce frontier $F=\{w \mid w$ consists of an equal number of a's and b's$\}$.

Hopcroft and Ullman(22) present a context-free phrase structure grammar that produces F. This grammar has start symbol S and productions:

$S \to aBS$

$S \to aB$

$S \to bAS$

$S \to bA$

$A \to bAA$

$A \to a$

$B \to aBB$

$B \to b$

Grammar G_1 has been formed so that it will produce, as its tree language, exactly the derivation trees generated by the Hopcroft and Ullman grammar, therefore the set of frontiers produced by G_1 must be exactly F.

Claim(2) Given regular u-tree grammar $G_2 = \langle \{S,A,B,a,b\}, \sigma_2', P_2, \{S\cdot\} \rangle$ over $\{S,A,B,a,b\}$ with P_2 ={the same set of productions as P_1 but we now regard them as u-tree productions, not tree productions} then $L_s(G_2) = F$. This is true since G_2 can produce every tree (which serves as a u-tree representation) that G_1 can, so $F \subseteq L_s(G_2)$. Furthermore each tree (a u-tree representation) it produces must contain the same symbols on its frontier as a tree in L(G), except the symbols may be in a different order. Any string in F, however, even with a permuted order, is still an element of F. Therefore $L_s(G_2) = F$.

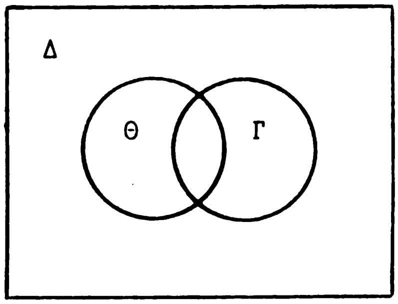
Now that Claim(2) is extablished we consider the language $F = L_s(G_2)$. It is easy to see (and well known) that F is not a regular set, but F is the frontier language of a regular u-tree grammar, G_2. So $\Gamma \not\subseteq \Theta$.

Q.E.D.

Lemma 5.5 $\Theta \cap \Gamma \neq \phi$.

Proof Let $D=\{0^n | n \geq 1\}$. $D \in \Theta$ and clearly $D \in \Gamma$. Q.E.D.

Theorem 5.5 The results of Lemmas 5.1, 5.2, 5.3, 5.4 and 5.5 can be summarized in the following Venn diagram of relationships.



We now investigate some of the closure properties of class Γ.

Theorem 5.6 Γ is closed under union.

Proof Let $A,B \in \Gamma$. Let $M_A = \langle C_A, \sigma_A'', P_A, \Gamma_A \rangle$ be defined over $\langle \Sigma_A, \sigma_A \rangle$ with $L_s(M_A)=A$. Let $M_B = \langle C_B, \sigma_B'', P_B, \Gamma_B \rangle$ be defined over $\langle \Sigma_B, \sigma_B \rangle$ with $L_s(M_B)=B$.

We will define a new grammar M such that $L_s(M)=A \cup B$, but in order to avoid interaction between productions in P_A and those in P_B we must first relabel some of the symbols.

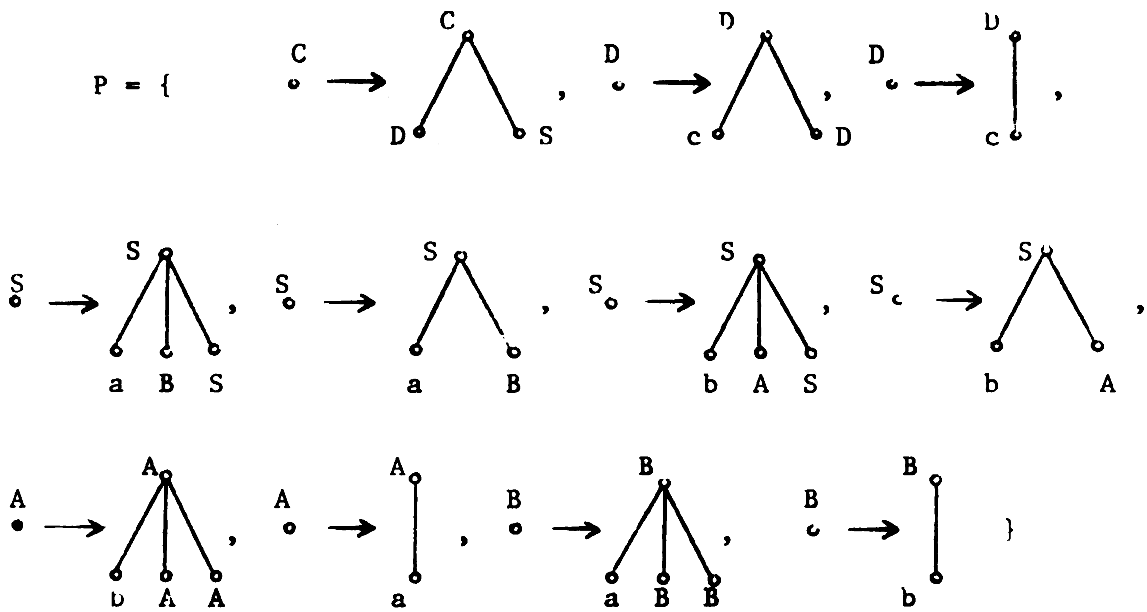
Let $C=C_A \cap C_B$. For each symbol $c_i \in C$ pick a unique new symbol $d_i \notin C_A \cup C_B$. Let $D=\{d_i | c_i \in C\}$. For all d_i, define $\sigma(d_i)=\sigma''(c_i)$. At each occurrence of c_i in P_B and in Γ_B replace c_i with d_i, call the new sets P_B'' and Γ_B'. Let $P_B'=P_B'' \cup \{d_i \cdot \rightarrow \cdot c_i | c_i \in C \cap \Sigma_B\}$. We now have a new machine $M_B' = \langle C_B'=C_B \cup D, \sigma_B'=\sigma_B'' \cup \{\sigma(d_i)\}, P_B', \Gamma_B' \rangle$ defined over $\langle \Sigma_B, \sigma_B \rangle$ with $L_s(M_B')=L_s(M_B)$.

Pick a new symbol $s \notin C_B' \cup C_A$. Let $M= \langle C_A \cup C_B' \cup \{S\}, \sigma_A'' \cup \sigma_B' \cup \{\sigma'(s)= 0\}, P_A \cup P_B' \cup \{s \cdot \rightarrow \psi | \psi \in \Gamma_A \cup \Gamma_B'\}, \{s \cdot\} \rangle$ be defined over $\langle \Sigma_A \cup \Sigma_B, \sigma_A \cup \sigma_B \rangle$.

Now, M will begin with start tree s• then immediately use a production producing a start tree in Γ_A or Γ_B'. Either way it will produce a string in $A \cup B$ and all such strings will appear. Q.E.D.

Lemma 5.6 Γ is not closed under intersection.

Proof Let $X=\{c^+w \text{ or } wc^+ | w$ consists of an equal number of a's and b's$\}$. Claim: $X \in \Gamma$. To see this consider regular u-tree grammar $G=\langle\{S,A,B,C,D,a,b,c\},\sigma,P,\{S\cdot\}\rangle$ defined over $\langle\{S,A,B,C,D,A,b,c\},\sigma\rangle$ with



So, (see the proof of Lemma 5.4) $L_s(G)=X \Rightarrow X\in\Gamma$ and our claim is established.

Let $Y=\{a^+w \text{ or } wa^+ | w$ consists of an equal number of b's and c's$\}$. In a manner similar to that above we can show that $Y\in\Gamma$.

We now proceed to show that $X \cap Y \notin \Gamma$. For notational purposes we let:

$\alpha_1=\{c^+w \text{ in set } X\}$

$\alpha_2=\{wc^+ \text{ in set } X\}$

$\beta_1=\{a^+w \text{ in set } Y\}$

$\beta_2=\{wa^+ \text{ in set } Y\}$

Then $\alpha_1 \cap \beta_1 = \phi$

$\alpha_1 \cap \beta_2 = \{c^n b^n a^n \mid n \geq 0\}$

$\alpha_2 \cap \beta_1 = \{a^n b^n c^n \mid n \geq 0\}$

$\alpha_2 \cap \beta_2 = \phi.$

Now $X \cap Y = (\alpha_1 \cup \alpha_2) \cap (\beta_1 \cup \beta_2) = (\alpha_1 \cap \beta_1) \cup (\alpha_1 \cap \beta_2) \cup (\alpha_2 \cap \beta_1) \cup (\alpha_2 \cap \beta_2)$ so $X \cap Y = \{c^n b^n a^n \mid n \geq 0\} \cup \{a^n b^n c^n \mid n \geq 0\}$ which is not even context-free (see Theorem 4.7, Hopcroft and Ullman(22)) therefore is not in Γ. Q.E.D.

Lemma 5.7 Γ is not closed under complementation.

Proof Assume Γ is closed under complementation. We also know it is closed under union, by Theorem 5.5. Now $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ therefore closure under union and complementation imply closure under intersection (x). Q.E.D.

Theorem 5.7 Γ is not a Boolean Algebra.

Proof Follows from Lemma 5.6 or 5.7.

CHAPTER 6

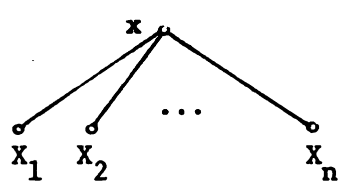
CHARACTERIZATIONS FOR TREE GRAMMARS

In this chapter a characterization for the classification of tree grammars along the same lines as that for phrase structure grammars is proposed. The work is intended to be exploratory; it provides a beginning towards a more general theory of tree grammars.

6.1 BASIS FOR CHARACTERIZATION

Theorem 6.1 The strings of labels encountered on the root-to-frontier paths in the trees of a language generated by an RTG will be a regular set.

Proof There will be a reduced, expansive RTG, G, that will generate any such tree language. Suppose $G = \langle B,\sigma',P,\{S\cdot\}\rangle$ over $\langle A,\sigma\rangle$. We define regular phrase structure $G' = \langle V_N = B-A, V_T = A, P', S\rangle$ where the productions in P' will be defined as follows:

For $\overset{X_0}{\circ} \longrightarrow$  in P, P' will contain:

$X_0 \to xX_1$
$X_0 \to xX_2$
.
.
.
$X_0 \to xX_n$

For $\overset{X_0}{\circ} \longrightarrow \overset{x}{\circ}$ in P, P' will contain:

$X_0 \to x.$

So G' will generate the strings of labels on root-to-frontier paths generated by G and G' is regular, therefore these strings form a regular set. Q.E.D.

The next theorem shows that for any phrase structure grammar, regardless of the type (3, 2, 1 or 0) productions, if each production is only applied to the right hand end of sentential forms, then only a regular set will be produced.

<u>Theorem</u> <u>6.2</u> Given phrase structure grammar, G. If the stipulation is made that for each production, p, in G with r symbols in its LHS, p can only be applied to the right hand r symbols of a sentential form generated by G, then L(G) is a regular set.

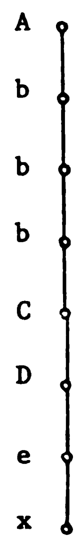
<u>Proof</u> Given grammar $G = \langle V_N, V_T, P, S \rangle$ we construct regular tree grammar $G' = \langle V_N \cup V_T, \sigma', P', \{S\cdot\} \rangle$ over $\langle V_T, \sigma \rangle$. For each production $x_1 x_2 \cdots x_n \rightarrow y_1 y_2 \cdots y_m$ in P we let

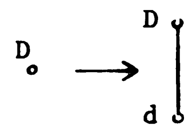
$$\begin{array}{c} x_1 \\ | \\ x_2 \\ \vdots \\ | \\ x_n \end{array} \longrightarrow \begin{array}{c} y_1 \\ | \\ y_2 \\ \vdots \\ | \\ y_m \end{array}$$

be in P'. G' is a regular tree grammar and it generates root-to-frontier paths in the same way that G generates strings. So {root-to-frontier paths generated by G'}=L(G), therefore, by Theorem 6.1, L(G) is regular. Q.E.D.

The following classification of regular (type 3), context-free (type 2), context-sensitive (type 1) and recursively enumerable (type 0) tree grammars seems appropriate. We will say a tree grammar is type i, for $0 \leq i \leq 3$, if every production in the grammar, when applied to a tree sentential form, has the effect on the strings formed by root-to-frontier paths of applying a type i phrase structure grammar production. As illustrated by Theorem 6.1, the formalization of regular (type 3) tree grammars fits the classification perfectly. When we write a formalization for type i grammars, with i<3, there is an added complication. Type 3 tree grammar productions are always applied at the end of a root-to-frontier path, at a place where there can be no sub-trees of the nodes involved which are themselves not involved in the production. When applying, say, a type 2 production it can, in general, be applied to a node in the interior of a root-to-frontier path, a node which may have several subtrees. The problem is also present in type 1 and type 0 tree productions, so each of these definitions must account for any subtrees which might be involved.

We want our definition of type i, for i<3, tree grammars to be a generalization of the type i phrase structure grammar in the sense that given a phrase structure sentential form, say AbbbCDex we may regard this as a tree with exactly one corresponding root-to-frontier path. The sentential form for the tree in this case will be:



When a context-free tree production, say 

is

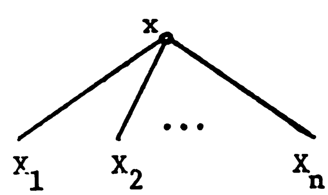
applied, it should have the same effect as applying D→Dd in the corresponding string which would leave AbbbCDdex. The subtree with root one level below D (i.e. $\begin{matrix} e \\ | \\ x \end{matrix}$) must be maintained one level below $\begin{matrix} D \\ | \\ d \end{matrix}$ giving:

$$\begin{matrix} D \\ | \\ d \\ | \\ e \\ | \\ x \end{matrix}$$

In general, subtrees whose roots occur one level below the node being replaced by a tree production must be retained on <u>at least</u> some of the root-to-frontier paths generated by applying a tree production. If they were not retained it would be equivalent to applying a production to a string sentential form generated by a context-free phrase structure grammar and eliminating everything to the right of the replacing symbols; this is clearly intolerable in any context-free grammar. With these considerations in mind we are ready to define context-free tree grammars.

6.2 CONTEXT-FREE TREE GRAMMARS

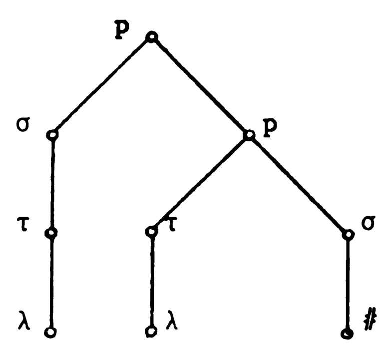
W.C. Rounds in (34) has proposed a good definition for context-free tree grammars. In order to discuss his definition we first informally introduce that class of tree transducers that was originally called generalized sequential machines (GSMs) by Thatcher(42) and Rounds(35) and has lately been termed top-down tree transducers and investigated by Baker(3), Englefreit(15), Ogden and Rounds(30) and Perrault(31) among others. We may think of a GSM as a device which serves as a mapping of trees over ranked alphabet, A, into trees over ranked alphabet, B, i.e. GSM: $T_A \to T_B$. A GSM is a finite-state device; its move function is a function from (old state,input) pairs (where each input has a certain number, the rank of the input, of arguments) into (new state,output) pairs with assignments of each of the input arguments into the output. The inputs will be nodes of trees, the arguments will be the subtrees of the input nodes, the output will be trees. A GSM works on a top-down basis, starting at the root of a tree in a given start state. We present an example from Rounds(34) to illustrate this informal discussion. In the following example and throughout the rest of this chapter we use the symbol



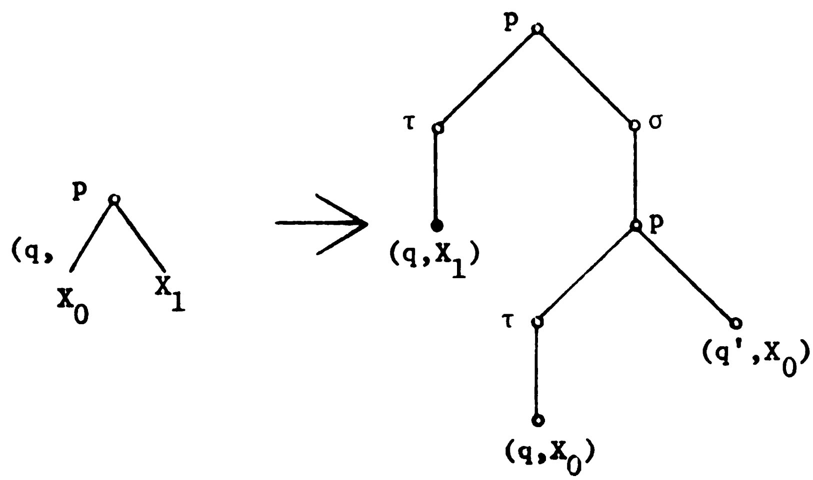
to indicate node x and the n distinct subtrees with roots one level below x. $X_1,\cdots,X_n$ are dummy arguments used to indicate the subtrees.

<u>Example 6.1</u> Both input and output alphabets are Σ. Let $\Sigma_2=\{p\}$, $\Sigma_1=\{\sigma,\tau\}$, $\Sigma_0=\{\#,\lambda\}$.

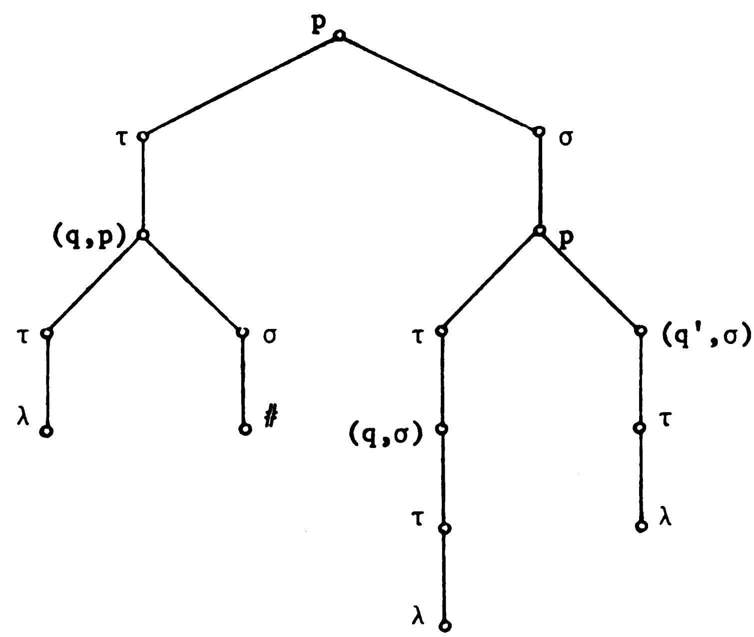
Then t =



is an element of T_Σ. If a GSM starts in start state q and has a transition function



defined, then the result on t will be:



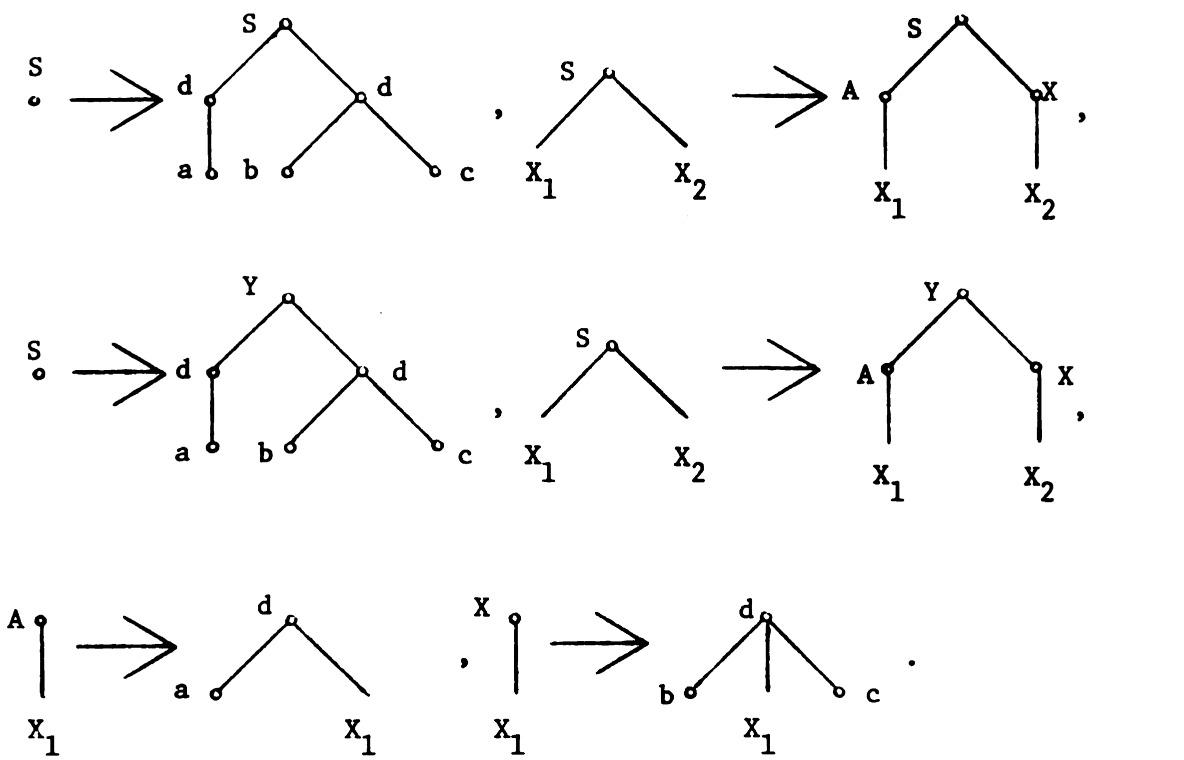
Other transition functions will then be applied on each of the (state, input pairs (q,p), (q,σ) and (q',σ).

Rounds uses a modification of the GSM definition as his definition of context-free tree grammar. Context-free tree grammars will map trees over a ranked alphabet into trees over the same alphabet. He regards (state,input) pairs as being nonterminals. He regards the transition function as actually defining a production. He liberalizes these productions so that nonterminal (state,input) pairs do not have to be on the frontier of the RHS of a production, rather they may be anywhere on the RHS but only one (or zero) per branch is permitted. Each subtree must appear below at least one of the new nodes.

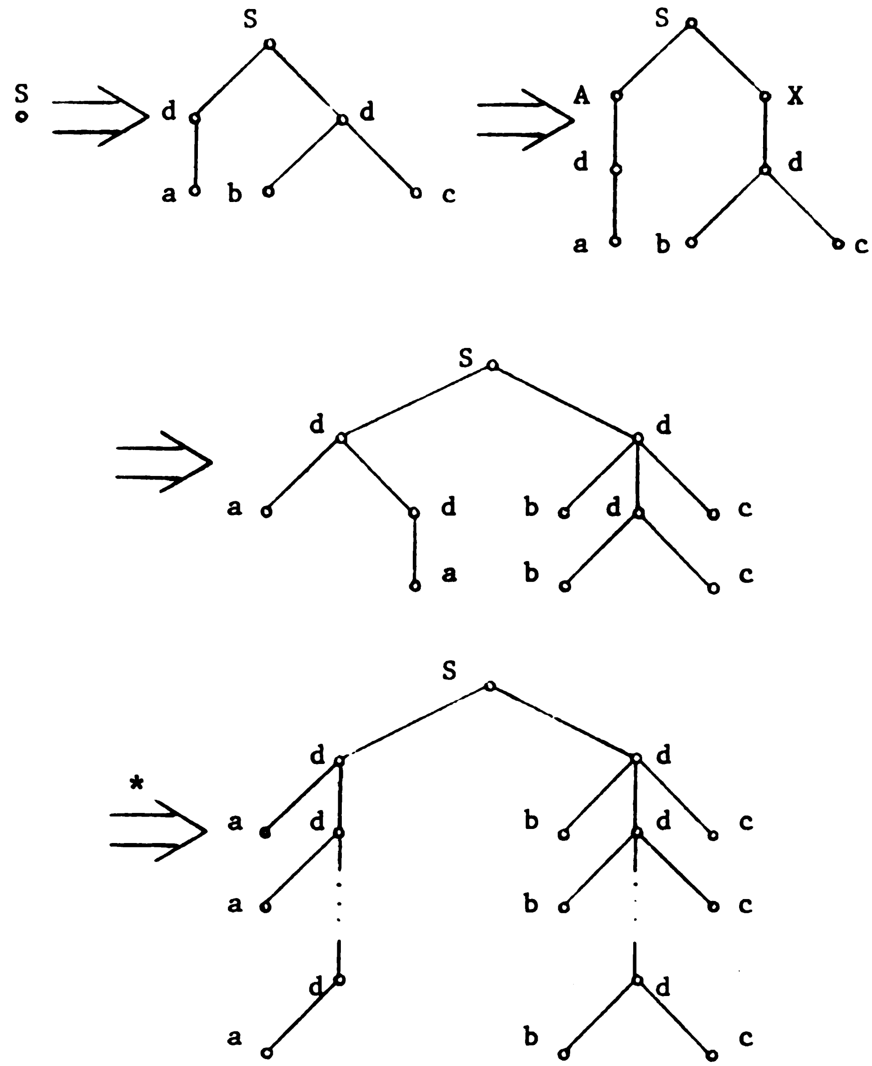
This definition seems more restrictive than desired in the sense that it only allows productions to be applied from the top to the bottom of the tree, i.e. from left to right on the root-to-frontier paths. This restriction is not important however, since we know that for every phrase structure context-free derivation there is an equivalent left to right context-free derivation.

Other definitions for context-free tree grammars are certainly possible. They should maintain the idea of applying context-free productions on root-to-frontier paths during a derivation. Other non-equivalent definitions, however, seem to be more restrictive in ways that restrict the power of productions and yet add nothing.

<u>Example</u> <u>6.2</u> Context-free tree productions can produce frontier $\{a^n b^n c^n | n \geq 1\}$ which is a context-sensitive language. Given productions:



Derivations will be of the form:



It is clear that context-free tree grammars, as just defined, are more general than regular tree grammars, yet every regular tree grammar (in expansive form) will be a context-free tree grammar. Furthermore, as illustrated by the previous example, the addition of context-free productions makes the grammars more powerful in terms of the frontiers they can produce.[1] It seems natural to ask whether the type 2 tree

---

[1]It was shown in Chapter 2 that the frontiers of languages generated by regular tree grammars are exactly the context-free languages, but the frontier generated in Example 6.2 was context-sensitive.

grammars may have exactly the type 1 string languages as their frontiers. The answer is no, the frontiers will be a proper subclass of the context-sensitive languages. To establish this fact we need a bit of background.

Indexed grammars were defined by A.V. Aho in (1). He showed that the class of languages generated by indexed grammars properly contains the class of context-free languages and is properly contained in the class of context-sensitive languages. We now restate a theorem from Rounds(34).

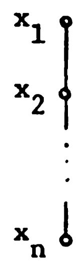
Theorem 6.3 (Rounds) For every context-free tree grammar, G, one may effectively find an indexed grammar, G', such that the frontier generated by G is exactly L(G').

Thus, although not every language generated by indexed grammars is the frontier generated by a context-free tree grammar, the converse is true and the class of these frontier languages must be properly contained in the context-sensitive languages.

6.3 CONTEXT-SENSITIVE TREE GRAMMARS

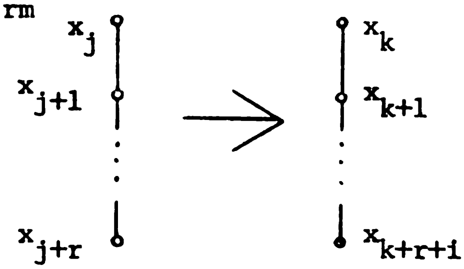
We now consider the question of what is an appropriate definition for context-sensitive tree grammars. We should include those that can apply context-sensitive string substitutions on the root-to-frontier paths in sentential form trees. Since there are several (probably equivalent) ways of defining these tree grammars we will again only describe them, rather than formally define them.

Given a string, say $x_1x_2\cdots x_n$, we must be able to apply a context-sensitive type substitution to it. If this string is thought of as a root-to-frontier path



we can apply, say $x_j x_{j+1} \cdots x_{j+r} \to x_k x_{k+1} \cdots x_{k+r+i}$ for $i \geq 0$.

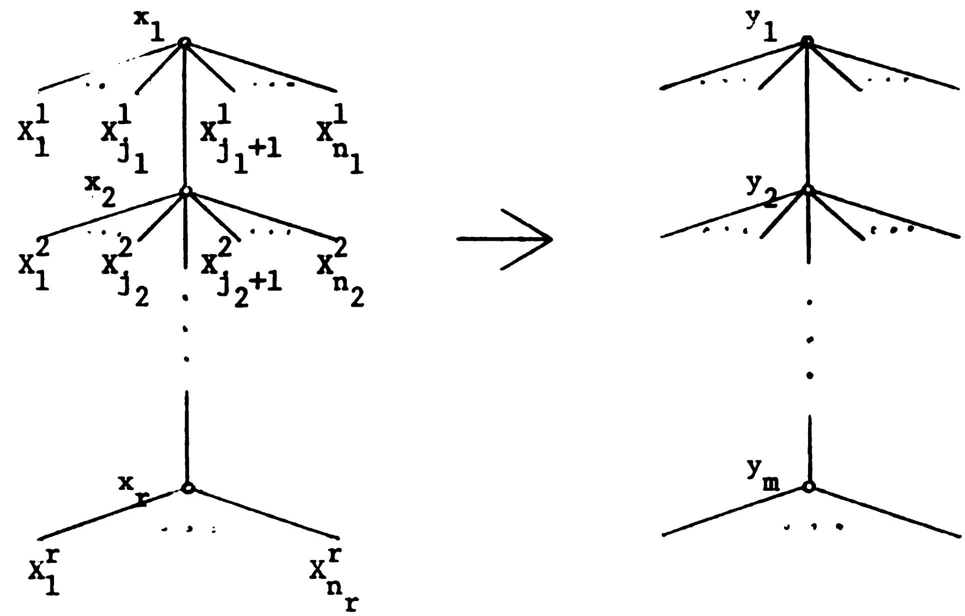
Thus productions of the form



must be allowed.

Also, some method of reassigning subtrees must be allowed. It is important, however, that no subtrees be assigned to nodes at a higher level than their original level; this would have the effect of shortening some root-to-frontier paths, i.e. applying distinctly non-context-sensitive productions.

So, the following description of possible context-sensitive tree productions seems appropriate. We use superscripts on dummy auguments to indicate what node they originally are subtrees of, and subscripts to indicate which subtree. Productions may be of the form: (see next page)



Each of the subtrees represented by $\{X_j^i \mid 1\leq i\leq r,\ 1\leq j\leq n_r\}$ must be duplicated at least once as a subtree of some y_k. Furthermore, the subtree represented by X_j^i must be duplicated on a y_k with $k\geq i$ in order to prevent shortening any root-to-frontier path. The shortening of strings can only occur with recursively enumerable productions, not context-sensitive.

Note that every context-free production will also be allowed as a context-sensitive production.

Example 6.3 The following context-sensitive tree grammar was constructed by first considering the context-sensitive phrase structure grammar with productions: $P' = \{$ S→aBSc

S→aBc

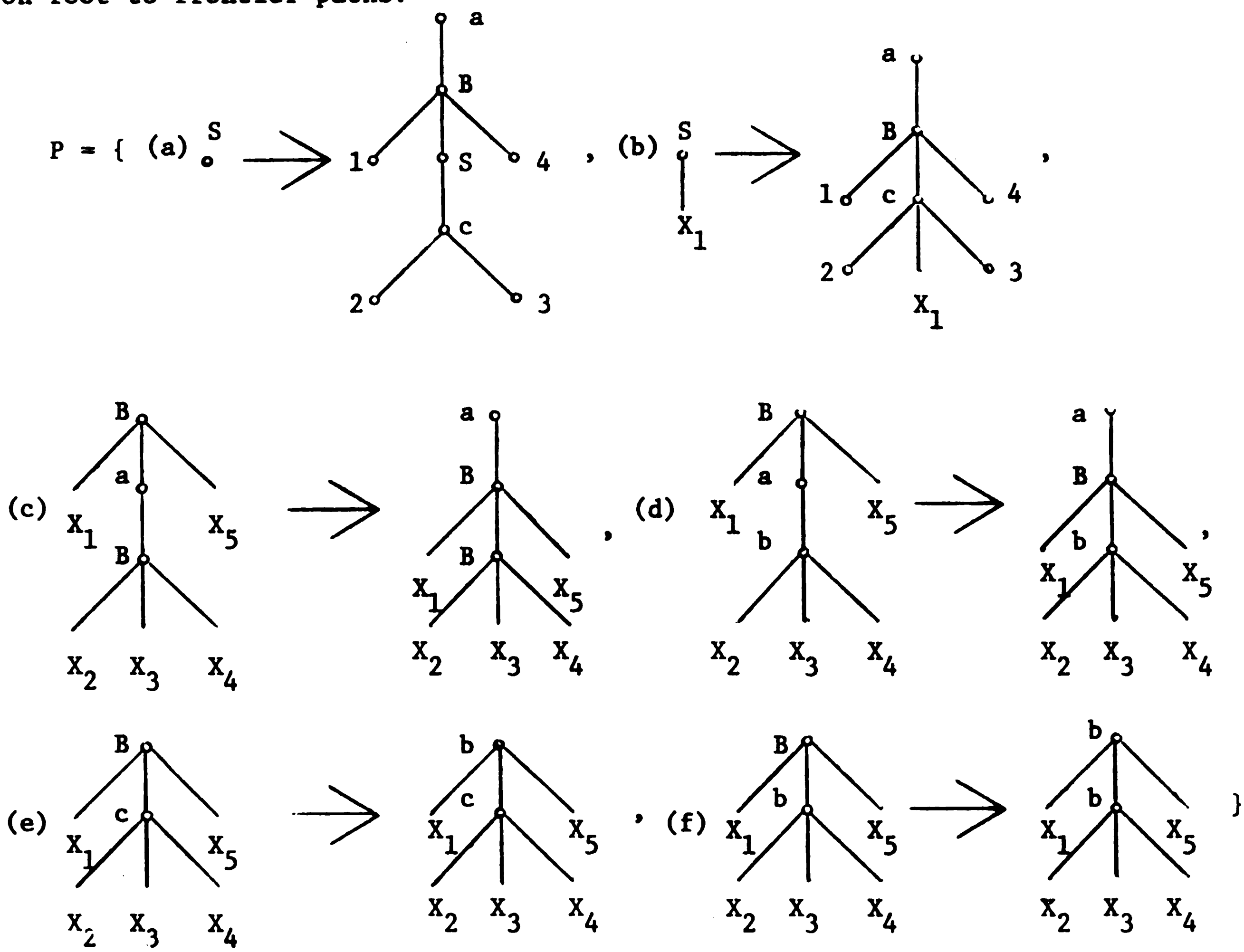
Ba→aB

Bc→bc

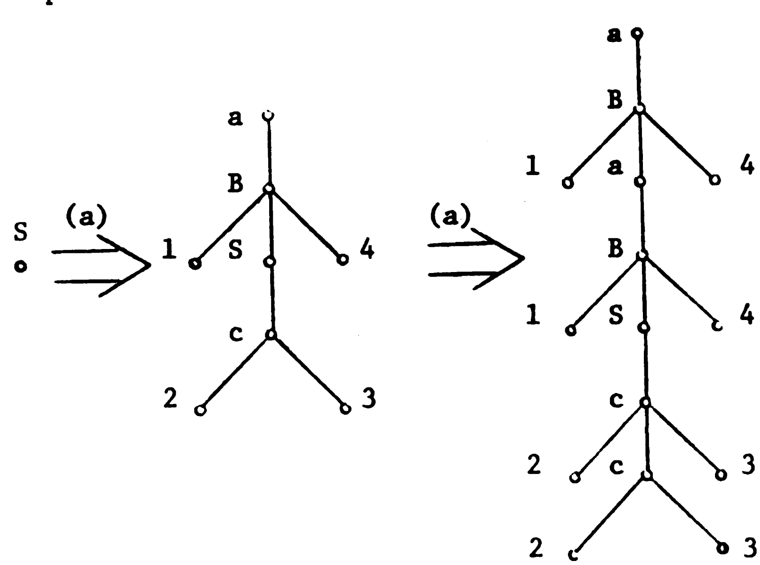
Bb→bb $\}$

These productions will give $\{a^n b^n c^n \mid n\geq 1\}$. We construct tree grammar $G = \left\langle \{1,2,3,4,a,b,c,S,B\},\sigma',P,\{S\cdot\}\right\rangle$ over $\left\langle \{1,2,3,4,a,b,c\},\sigma\right\rangle$.

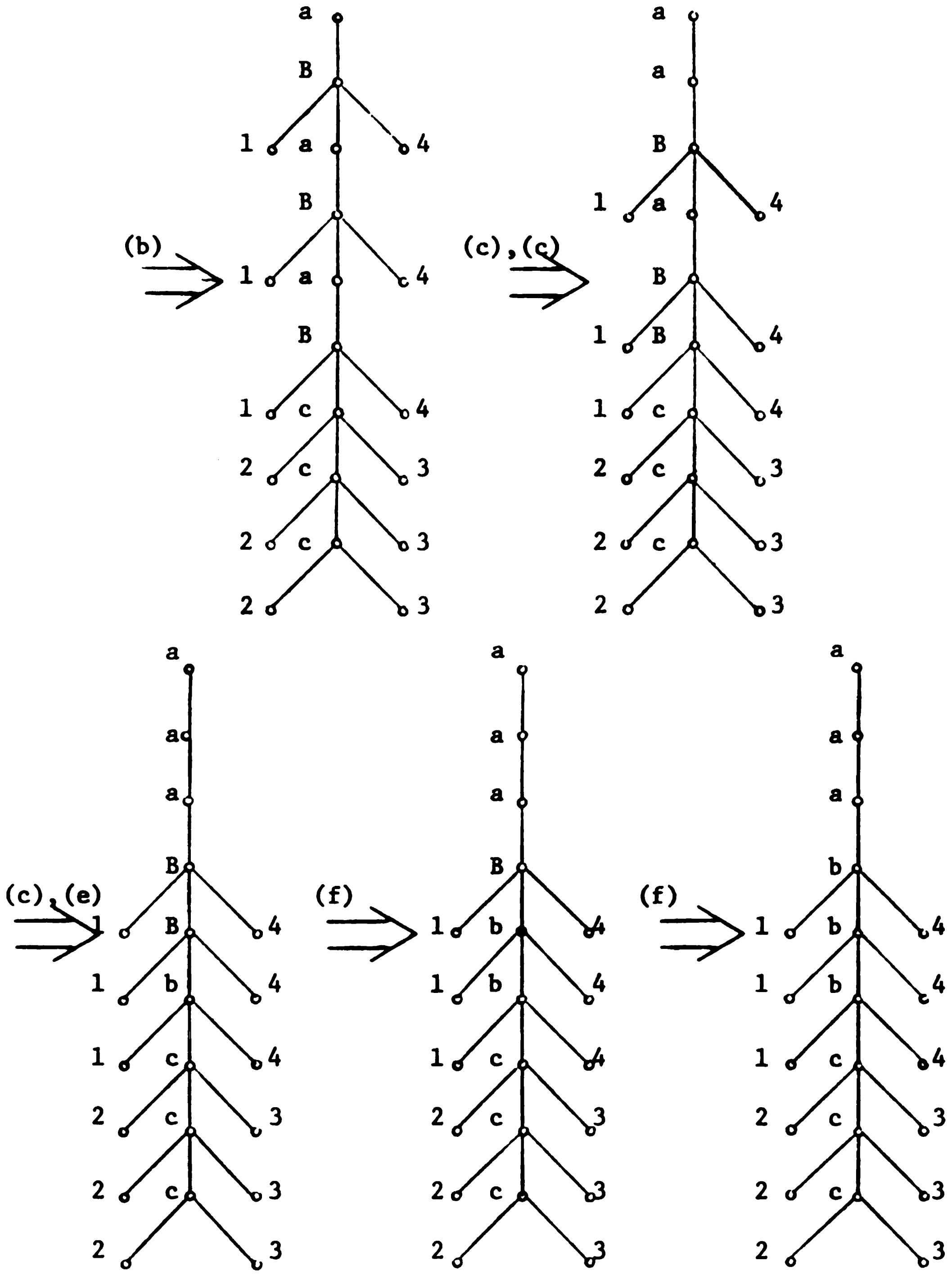
The productions in P will essentially model those in P', when applied on root-to-frontier paths.



A sample derivation is:



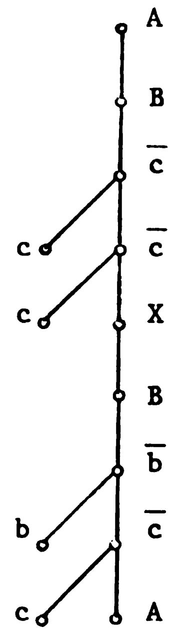
(continued next page)



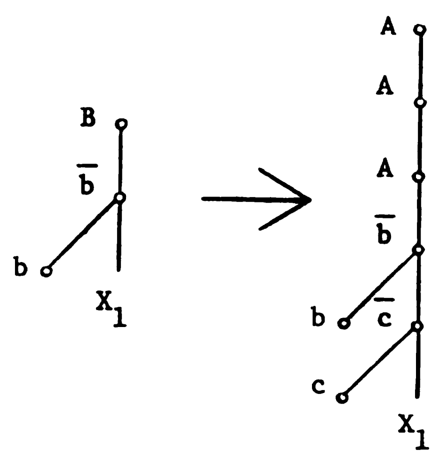
In general the frontier of the trees produced will be $\{1^n 2^n 3^n 4^n | n \geq 1\}$.

Context-sensitive (and recursively-enumerable) tree grammar concepts may be applicible to more general graph grammars. This is discussed in Chapter 7.

One might ask what class of languages will be generated as the frontiers of context-sensitive tree productions. Although the complete answer is not known at this time it is easy to see that at least all the context-sensitive languages can be generated. If we have a context-sensitive phrase structure grammar which produces sentential form ABccXBbcA we may represent this as the following tree:



If the context-sensitive grammar has production Bb→AAAbc we may include corresponding tree production:



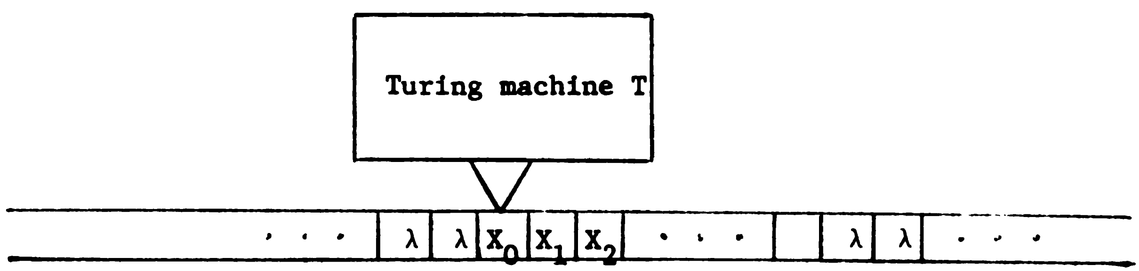
Productions of this type will eventually produce frontier

$\alpha_1\alpha_2\cdots\alpha_n$ for a similar string $\alpha_1\alpha_2\cdots\alpha_n$ generated by the phrase structure grammar.

6.4 RECURSIVELY ENUMERABLE TREE GRAMMARS

The definition of type 0 tree grammars should be the same as that for type 1 except that here it will be legitimate for subtrees to be assigned (by a production application) to a node at a lower level (nearer the root) than where they occurred before the production was applied.

Example 6.4 We examine an interesting recursively enumerable grammar, one that can completely model a Turing machine.[2] We assume the Turing machine to be modeled is in the following form:



In the starting configuration the read head is on the leftmost input tape cell. The input, or starting tape configuration, is $X_0,X_1,\cdots,X_n$. Symbols to the left of X_0 are null input symbols, as are those to the right of X_n. We assume T starts in state s_0. The output/move function for T will be stated as a partial function with finite domain of the form: $\delta:S\times I \to S\times O\times M$ with:

---

[2]The idea for modeling a Turing machine with a tree grammar was suggested to me in a personal conversation with W.C. Rounds.

S = states of T

I = read symbols (the input symbols)

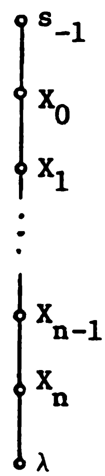
O = output symbols

M = {L,N,R} L means the read head will move to the left one cell on the tape, R means it will move to the right one cell and N means no move.

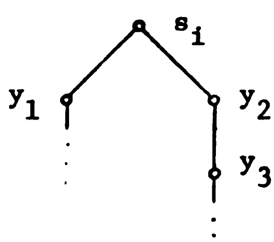
If no move function is defined for a (state,input) pair, T halts.

Since for every nondeterministic Turing machine there is an equivalent deterministic one, we assume WOLOG that T is deterministic.

The start tree for our grammar, G, will be based on a starting configuration for T. This will be:

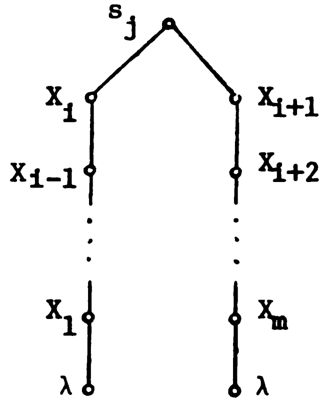


The alphabet for G will be $I \cup O \cup S \cup \{s_{-1}\}$. Productions in G will simulate δ in the following manner:

Tree 

may be thought of as T in state s_i with the read head on cell containing y_2 with y_1 in the cell to the left and y_3 in the cell to the right. G (after applying one extra production to get started) will simulate T by allowing exactly one production to be applied at each point where T would have a δ function applied. If T halts on input $X_0 \cdots X_n$, G will have no further productions possible to apply.

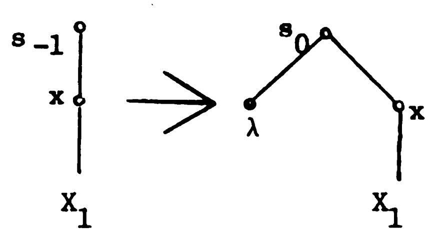
If we have tree



produced by G with no further applicible productions then the final tape configuration for T will be given by $X_1X_2 \cdots X_{i-1}X_iX_{i+1} \cdots X_m$ with the read head on cell containing X_{i+1}. If T does not halt on input $X_0 \cdots X_n$ then G will never cease to have productions to apply, and after applying the (i+1)th production the tape configuration for T after its (i)th move function can be read by the procedure described above.

Now to define the productions in G. First the set of productions to "get started". Define:

P_A:

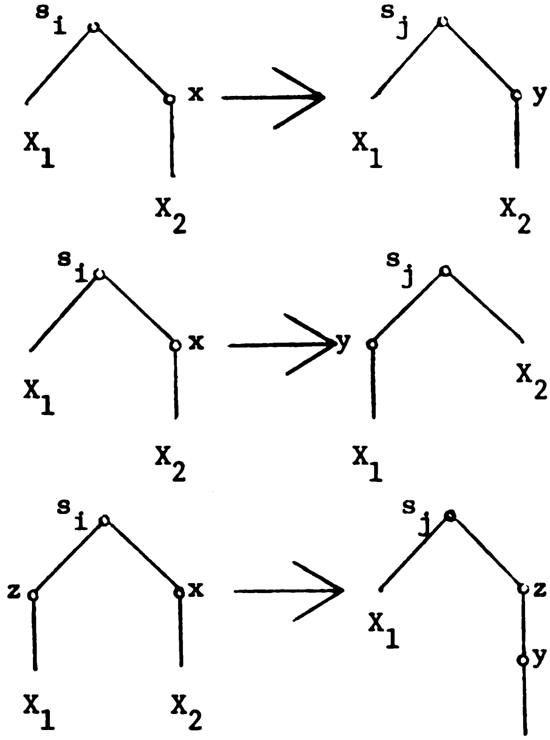


for all x in I.

Exactly one production from P_A will be applied at the beginning of each derivation in G.

Now we define the productions that simulate δ:

P_B:



For all $\delta(s_i,x)=(s_j,y,N)$ in T.

For all $\delta(s_i,x)=(s_j,y,R)$ in T.

For all z in $O \cup I$,
For all $\delta(s_i,x)=(s_j,y,L)$ in T.

It is easy to see that a production from P_B can only be applied if an equivalent move function in T would be applied.

Note that in productions in P_B we are taking advantage of the fact that recursively enumerable tree productions allow the shortening of root-to-frontier paths.

The frontier languages generated by type 0 tree grammars are exactly the type 0 languages. By altering our Turing machine simulation to let the terminal symbols hang one level below place holder nodes, as was discussed for context-sensitive tree productions, we can generate any type 0 language generated by a Turing machine.

CHAPTER 7

SUMMARY AND RECOMMENDATIONS

7.1 SUMMARY

The study of multidimensional automata, specifically tree automata theory, holds a great deal of promise for automatic pattern recognition. The first use of tree automata for this purpose was shown by Fu and Bhargava in (20). This thesis uses concepts from graph theory to provide significant extensions of (20) in several directions. It also shows a number of theoretical properties of tree grammars and automata. It provides another step towards developing automatic methods for manipulating graphs.

Chapter 1 presents a general introduction to the pattern recognition problem and to the study of multidimensional automata. A brief survey of applications of multidimensional automata is included.

Chapter 2 summarizes the pertinent work of other researchers in tree automata theory. The rest of this thesis draws heavily on the theoretical bases provided here. Most of this theory was developed by Brainerd and Thatcher.

In Chapter 3 the graph based structure for u-trees and d-trees is developed. Regular u-tree grammars and u-tree automata are defined and their relationships are shown. The various properties of these systems are investigated.

In Chapter 4 we show how these grammars and automata can be effectively used to provide an automatic (once the primitives are found) method for pattern recognition. Many of the terms and concepts inherent in any syntactic pattern recognition scheme are formalized here for the first time. Important differences between circular and noncircular primitive systems are noted. A partial characterization for sets of patterns recognizable by this system is given. Several examples and special cases are worked.

Chapter 5 deals with the question of simplifying the machines to classify patterns. It is found that a deterministic pushdown automaton will suffice rather than a u-tree automaton. This makes the pattern recognition scheme a more practical method since deterministic pushdown automata are easy to simulate using digital computers. Also in this chapter we explore the properties of the frontier languages generated by regular u-tree grammars.

In Chapter 6 a basis for providing a characterization for various types of tree generating grammars is proposed. Regular, context-free, context-sensitive and recursively enumerable grammars are described using this characterization and the frontier languages generated are investigated.

7.2 SUGGESTIONS FOR FUTURE WORK

First we discuss a very general problem. This thesis has developed a graph generation and classification scheme for certain kinds of acyclic rooted digraphs. Some of the same methods (particularly the generative ones) should be applicible to graphs in general. Consider, in particular, the recursively enumerable tree grammars described in

Chapter 6. What would be the result of applying generative procedures of this form to unrooted graphs (with appropriate conventions as to what paths might replace root-to-frontier paths)? How about applying context-free productions? The automata defined here depend heavily on the ability to conveniently represent tree-like structures in linear form and on their being rooted. What kinds of automata might one define that do not use these properties? The investigation of questions along these lines seems worthwile.

Another problem that arises with any application involving formally defined grammars is the question of grammatical inference: how can an appropriate grammar be developed from the observation of sentences in a language? In the present case the problem is twofold: given a pattern set of interest we need to find an appropriate primitive system to represent the patterns as well as an appropriate regular u-tree grammar using the primitive types. What sort of procedures might one carry out in order to define these?

It is possible that u-tree grammars and u-tree automata may be good formalizations to represent the growth and manipulation of and/or decision trees and search trees of the type shown in Nilsson(29). The idea of unorderdness is inherent in these trees so perhaps something like this would be fruitful. Some relaxing of the u-tree grammar definition might be necessary in order to allow for an infinite number of node labels in such applications.

In the various theorems and results presented in Chapter 4 we have partially characterized the sets of patterns that can be recognized by the pattern recognition method outlined. If this method proves practical for pattern recognition applications perhaps a more complete

characterization can be developed.

One of the practical considerations that has hindered the application of syntactic techniques to real pattern recognition problems is the noise and imperfect data that arises with real data. Ellis in (14) has proposed several models for probabilistic tree automata. Perhaps a useful probabilistic model could be constructed for u-tree grammars and automata so that small imperfections in data could be dealt with.

Also as a practical consideration the deterministic pushdown automaton construction outlined in Theorem 5.4 produces a very large PDA. Many of its move function rules are redundant and/or unnecessary. It may be possible to refine it somewhat, i.e. to simplify the move function.

The context-free, context-sensitive and recursively enumerable tree grammars described in Chapter 6 immediately give rise to the question of what types of automata might be appropriate to recognize the tree languages generated. Just as regular tree grammars and pseudoautomata can deal with the same sets of trees there are probably distinct automata models that provide acceptors for each of the classes of tree languages generated by these grammars.

The frontier languages generated by each of these types of tree grammars is shown in the following table:

Type of Tree Grammar	Type Frontier Language Generated
Regular	Context-free
Context-free	Indexed
Context-sensitive	? (at least context-sensitive)
Recursively enumerable	Recursively enumerable

It would be interesting to discover exactly what class of frontier languages can be generated by the context-sensitive tree grammars.

BIBLIOGRAPHY

BIBLIOGRAPHY

(1) Aho, A.V., "Indexed Grammars - an Extension of Context-Free Grammars", IEEE Conference Record, Symposium on Switching and Automata Theory, pp. 21-31, 1967.

(2) Aho, A.V. and J.D. Ullman, "Translations on a Context-Free Grammar", Conference Record of First ACM Symposium on Theory of Computing, pp. 93-112, 1969.

(3) Baker, B.S., "Tree Transductions and Families of Tree Languages: Abstract", to be included in the author's Ph.D. dissertation, Harvard University (in preparation).

(4) Blum, M. and C. Hewitt, "Automata on a 2-Dimensional Tape", IEEE Conference Record of Eighth Annual Symposium on Switching and Automata Theory, pp. 155-160, 1967.

(5) Brainerd, W.S., "The Minimilization of Tree Automata", Information and Control, Vol. 13, pp. 484-491, 1968.

(6) Brainerd, W.S., "Semi-Thue Systems and Representations of Trees", IEEE Conference Record of Tenth Annual Symposium on Switching and Automata Theory, pp. 240-244, 1969.

(7) Brainerd, W.S., "Tree Generating Regular Systems", Information and Control, Vol. 14, pp. 217-231, 1969.

(8) Buchi, J.R. and C.C. Elgot, "Decision Problems of Weak Second-Order Arithmetics and Finite Automata", Abstract 553-112, Notices Amer. Math. Soc., Vol. 5, p. 834, 1958.

(9) Buchi, J.R., "Weak Second-Order Arithmetic and Finite Automata", University of Michigan, Logic of Computers Group Technical Report, September 1959; Z. Math. Logik Grundlagen Math., Vol. 6, pp. 66-92, 1960.

(10) Buttleman, H.W., "On Generalized Finite Automata and Unrestricted Generative Grammars", Proceedings of Third Annual ACM Symposium on Theory of Computing, pp. 63-77, 1971.

(11) Chomsky, N., "Aspects of the Theory of Syntax", The M.I.T. Press, 1965.

(12) Doner,J.E., "Decidability of the Weak Second-Order Theory of Two Successors", Abstract 65T-468, Notices Amer. Math. Soc., Vol. 12, p. 819, 1965; erratum, ibid, Vol. 13, p. 513, 1966.

(13) Elgot,C.C., "Decision Problems of Finite Automaton Design and Related Arithmetics", University of Michigan, Dept. of Mathematics and Logic of Computers Group Technical Report, June 1959; Trans. Amer. Math. Soc., Vol. 98, pp. 21-51, 1961.

(14) Ellis,C.A., "Probabilistic Tree Automata", Information and Control, Vol. 19, pp. 401-416, 1971.

(15) Engelfriet,J., "Bottomup and Topdown Tree Transformations--A Comparison", Tech. Report, Technische Hogeschool Twente, Netherlands, July, 1971.

(16) Fischer,M.J., "Two Characterizations of the Context-Sensitive Languages", IEEE Conference Record of Eighth Annual Symposium on Switching and Automata Theory, pp. 149-156, 1969.

(17) Fu,K.S., "On Syntactic Pattern Recognition and Stochastic Languages", Purdue University School of Electrical Engineering Tech. Report No. 71-21, January, 1971.

(18) Fu,K.S. and P.H. Swain, "On Syntactic Pattern Recognition", Software Engineering, Vol. 2, J.T. Tou editor, New York, Academic Press, pp. 155-183, 1971.

(19) Fu,K.S. and T. Huang, "Stochastic Grammars and Languages", Int. Journal of Computer and Info. Sciences, Vol. 1, No. 2, pp. 135-169, June, 1972.

(20) Fu,K.S. and B.K. Bhargava, "Tree Systems For Syntactic Pattern Recognition", Presented at the Computer Image Processing and Recognition Symposium, University of Missouri - Columbia, August 24-26, 1972.

(21) Harary,F., "Graph Theory", Addison-Wesley Publishing Company, 1969.

(22) Hopcroft,J.E. and J.D. Ullman, "Formal Languages and Their Relation to Automata", Addison-Wesley Publishing Company, 1969.

(23) Knuth,D.E., "Fundamental Algorithms, Vol. 1, The Art of Computer Programming", Addison-Wesley Publishing Company, 1968.

(24) Levy,L.S., "Tree Adjunct, Parenthesis and Distributed Adjunct Brammars", Theory of Machines and Computations, edited by Z. Kohavi and A. Paz, Academic Press, 1971.

(25) Levy,L.S. and A.K. Joshi, "Some Results in Tree Automata", Proceedings of Third Annual ACM Symposium on Theory of Computing, pp. 78-83, 1971.

(26) Martin,D.F. and S.H. Vere, "On Syntax-Directed Transduction and Tree Transducers", Second Annual ACM Symposium on Theory of Computing, pp. 129-135, 1970.

(27) Meyers,W.J., "Linear Representation of Tree Structure", Proceedings of Third Annual ACM Symposium on Theory of Computing, 1971.

(28) Miller,W.F. and A.C. Shaw, "Linguistic Methods in Picture Processing - A Survey", AFIPS Conference Proceedings, Vol. 33, Part 1, FJCC, 1968.

(29) Nilsson,N.J., "Problem - Solving Methods in Artificial Intelligence", McGraw-Hill Computer Science Series, 1971.

(30) Ogden,W.F. and W.C. Rounds, "Compositions of n Tree Transducers", Proceedings of Fourth Annual ACM Symposium on Theory of Computing, pp. 198-206, 1972.

(31) Perrault,C.R., "Tree Stack Transducers and Finite State Target Languages", University of Michigan Dept. of Computer and Communication Sciences Tech. Report No. M-26, March, 1973.

(32) Peters,P.S. and R.W. Ritchie, "Context-Sensitive Immediate Constituent Analysis: Context-Free Languages Revisited", Proceedings ACM Symposium on Theory of Computing, pp. 109-116, 1969.

(33) Pfaltz,J.L. and A. Rosenfeld, "Web Grammars", Proceedings Joint International Conference on Artificial Intelligence, Washington, D.C., pp. 609-619, 1969.

(34) Rounds,W.C., "Context-Free Grammars on Trees", ACM Symposium on Theory of Computing, pp. 143-148, 1969.

(35) Rounds,W.C., "Mappings and Grammars on Trees", Mathematical Systems Theory, Vol. 4, No. 3, pp. 257-287, 1970.

(36) Rounds,W.C., "Tree-Oriented Proofs of Some Theorems on Context-Free and Indexed Languages", Second Annual ACM Symposium on Theory of Computing, pp. 109-116, 1970.

(37) Savitch,W.J., "Maze Recognizing Automata", Proceedings of Fourth Annual ACM Symposium on Theory of Computing, pp. 151-156, 1972.

(38) Shaw,A., "A Formal Picture Description Scheme as a Basis for Picture Processing Systems", Information and Control, Vol. 14, pp. 9-52, 1969.

(39) Thatcher, J.W., "Characterizing Derivation Trees of Context-Free Grammars through a Generalization of Finite Automata Theory", Journal of Computer and System Sciences, Vol. 1, No. 4, pp. 317-322, 1967.

(40) Thatcher, J.W., "Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic", Mathematics Systems Theory, Vol. 2, No. 1, pp. 57-81, 1968.

(41) Thatcher, J.W., "Transformations and Translations from the Point of View of Generalized Finite Automata Theory", Conference Record ACM Symposium on Theory of Computing, pp. 129-142, 1969.

(42) Thatcher, J.W., "Generalized^2 Sequential Machine Maps", Journal of Computer and System Sciences, Vol. 4, No. 4, pp. 339-367, 1970.

(43) Thatcher, J.W., "Tree Automata - An Informal Survey", Unpublished paper, 1970.