JOURNAL OF COMPUTER AND SYSTEM SCIENCES 30, 25–40 (1985)

Tree Pushdown Automata

KARL M. SCHIMPF

Department of Computer and Information Sciences, University of California, Santa Cruz, California

AND

JEAN H. GALLIER

Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania 19104

Received March 5, 1983; revised June 1, 1984

This paper presents a new type of automaton called a tree pushdown automaton (a bottom-up tree automaton augmented with internal memory in the form of a tree, similar to the way a stack is added to a finite state machine to produce a pushdown automaton) and shows that the class of languages recognized by such automata is identical to the class of context-free tree languages.

I. INTRODUCTION

Tree languages are frequently used to model computations based on tree (or hierarchical) structures. For example, one can use tree structures to model the calling structure of a set of procedures and functions such as in derivation trees, syntax directed translation, and recursive schemes [30, 2, 15, 16, 8, 32, 6, 7, 17, 25]. These applications of tree language theory frequently use the methods and results of formal language theory [27, 28, 9, 10, 31]. In particular, these applications can be characterized using context-free tree languages [27, 28, 12, 11, 7]. Other applications include the class of macro (or indexed) languages since the class of languages generated by the yield of context-free tree languages is the class of macro (context-sensitive) string languages [1, 13, 14, 25, 9, 11, 26, 27].

The grammatical [9, 13, 14, 28] and algebraic [10, 11, 26] aspects of context-free tree languages have been explored fairly extensively and comprehensive accounts can be found in [10, 11]. On the other hand the machinery (in particular, the type of automaton) needed to recognize any particular language in this class of languages has not been studied except for recent concurrent discoveries by Guessarian [21, 22] and Schimpf [29], who have defined different types of tree pushdown automata. The difference between the two types of models (like tree automata) is the direction the tree is parsed (top-down by Guessarian and bottom-up by Schimpf).

0022-0000/85 $3.00

This paper presents the type of tree pushdown automaton investigated by Schimpf [29] and shows that the class of languages recognized by such automata is identical to the class of context-free tree languages. In this model of a tree pushdown automaton, a tree (called a tree stack) is associated with each read-head of the bottom-up tree automaton, to remember the structure of the input tree already scanned. The automaton uses this information as a guide to parse the remaining unscanned portions of the input tree.

This paper is organized as follows: Section II begins by presenting the definition of a Σ-tree and the operators on trees consisting of the notions of subtrees, tree replacement, and tree rewrites. Section III presents the definition of context-free tree grammars and context-free tree languages, including a normal form for context-free tree grammars known as Chomsky normal form. Section IV presents the definition of a tree pushdown automaton, and the language accepted by such an automaton. Section V shows the equivalence between the class of context-free tree languages and the class of languages accepted, by the tree pushdown automata, by showing methods of converting context-free grammars to pushdown automata and conversely. Finally, Section VI provides some concluding remarks.

II. Preliminary Notions

2.1. *Ranked Alphabets*

A *ranked alphabet* (sometimes called a stratified or graded alphabet) is a pair (Σ, r) consisting of an alphabet Σ and a rank function $r: \Sigma \to \mathbf{N}$ (where $\mathbf{N}$ denotes the set of nonnegative integers). Every symbol σ in Σ such that $r(\sigma) = n$ is said to have arity (or rank) n. Symbols in Σ are called *function symbols*, where the arity denotes the number of parameters the function has and symbols with arity zero are also called *constants*. For notational convenience, a ranked alphabet (Σ, r) will simply be denoted by Σ.

2.2. *Tree Domains*

A *tree domain* $\mathbf{D}$ [19, 20] is a nonempty set of strings over the set of positive integers $\mathbf{N}_+$ satisfying the following two conditions:

(i) For all d in $\mathbf{D}$, every prefix of d is also in $\mathbf{D}$.

(ii) For all d in $\mathbf{D}$ and every integer i in $\mathbf{N}_+$, if $d \cdot i$ is in $\mathbf{D}$, then for all j in $\mathbf{N}_+$ such that $1 \leqslant j \leqslant i$ the string $d \cdot j$ is also in $\mathbf{D}$.

Note that a tree domain provides an addressing scheme which uniquely identifies each node within a tree. This is achieved by denoting the root of the tree by the empty string ε and the ith descendant of a node d by the string $d \cdot i$.

2.3. *Σ-Trees*

Given a ranked alphabet Σ, a *Σ-tree* (or tree for short) is a function $t: D \to \Sigma$ such that

(i) $\mathbf{D}$ is a tree domain.

(ii) For all d in $\mathbf{D}$, if $n = |\{i \mid i \in \mathbf{N}_+, d \cdot i \in \mathbf{D}\}|$ then $n = r(t(d))$.

The domain of a tree t is denoted as $\mathrm{dom}(t)$ and elements of $\mathrm{dom}(t)$ are called *tree addresses.* A *node* is a pair (d, σ) in $(\mathbf{D} \times \Sigma)$, where d is a tree address in $\mathbf{D}$ and $\sigma = t(d)$. A node (d, σ) is a *leaf* if $r(\sigma) = 0$, otherwise (d, σ) is an *internal node.* The set of *leaf nodes*, denoted $\mathrm{leaf}(t)$, is the set $\mathrm{leaf}(t) = \{(d, \sigma) \mid (d, \sigma) \in t, r(\sigma) = 0\}$. Furthermore, given any ranked alphabet Σ, the set of all finite Σ-trees is denoted as $\mathbf{T}_\Sigma$.

2.4. *Trees with Variables*

Let $\mathbf{X}_n$ denote a set of n variables, where $\mathbf{X}_n = \{x_1, ..., x_n\}$ (and $X_0 = \varnothing$). Adjoining $\mathbf{X}_n$ to the set of constants of a ranked alphabet Σ, one obtains a set of all finite Σ-trees with variables in $\mathbf{X}_n$ which is denoted as $\mathbf{T}_\Sigma(\mathbf{X}_n)$ (note that $\mathbf{T}_\Sigma = \mathbf{T}_\Sigma(\mathbf{X}_0)$). Furthermore, given any tree $t \in \mathbf{T}_\Sigma(\mathbf{X}_n)$, let $\mathrm{var}(t)$ denote the set of tree addresses in t labeled by variables in $\mathbf{X}_n$. That is, $\mathrm{var}(t) = \{d \mid (d, x_i) \in t, x_i \in \mathbf{X}_n\}$. For notational convenience, variables x_1 through x_4 will frequently be denoted by x, y, z, and w, respectively.

2.5. *Subtrees*

Given a tree t and a tree address d in $\mathrm{dom}(t)$, the *subtree rooted at d in t*, denoted t/d, is the tree defined by the function consisting of the set of ordered pairs $\{(d', t(d \cdot d')) \mid d \cdot d' \in \mathrm{dom}(t)\}$.

2.6. *Tree Replacement*

Given two trees t_1, t_2 and a tree address d in $\mathrm{dom}(t_1)$, the *replacement* of t_2 *for the subtree* t_1/d, denoted $t_1[d \leftarrow t_2]$, is the tree defined by the function consisting of the set of ordered pairs

$$\{(d', t_1(d')) \mid d' \in \mathrm{dom}(t_1), d \text{ is not a prefix of } d'\}$$
$$\cup \{(d \cdot d', t_2(d')) \mid d' \in \mathrm{dom}(t_2)\}.$$

2.7. *Tree Composition*

Let Σ be a ranked alphabet. Given any tree $t \in \mathbf{T}_\Sigma(\mathbf{X}_n)$ and a sequence of trees $t_1, ..., t_n \in \mathbf{T}_\Sigma(\mathbf{X}_m)$, the *composition* (or tree addition) of the function t with functions t_1 through t_n is the tree defined by the set ordered pairs

$$\{(d, \sigma) \mid (d, \sigma) \in t, \sigma \in \Sigma\} \cup \{(d \cdot d', \sigma) \mid (d', \sigma) \in t_i, (d, x_i) \in t\}.$$

In other words, all occurrences of the variable x_i, $1 \leqslant i \leqslant n$, in the tree t, are simultaneously replaced by the tree t_i. Let the composition of t with trees $t_1, ..., t_n$ be denoted as $t(t_1, ..., t_n)$.

2.8. *Rewrite Rules*

Let Σ be a ranked alphabet and n a constant in $\mathbf{N}$. A set $\mathbf{P}$ of *rewrite rules* (or productions) is a finite set of ordered pairs of trees of the form $(t_1, t_2) \in \mathbf{T}_\Sigma(\mathbf{X}_n) \times \mathbf{T}_\Sigma(\mathbf{X}_n)$. Each pair (t_1, t_2) in $\mathbf{P}$ is called a *rewrite rule* (or production) and is denoted as $t_1 \to t_2$. Given a set of rewrite rules $\mathbf{P}$ and two trees $s_1, s_2 \in \mathbf{T}_\Sigma(\mathbf{X}_n)$, s_1 *rewrites* to s_2 (denoted $s_1 \Rightarrow_{\mathbf{P}} s_2$) iff there exists a tree $t \in \mathbf{T}_\Sigma(\mathbf{X}_n)$ and a rewrite rule $t_1 \to t_2 \in \mathbf{P}$ such that

$$s_1 = t[d \leftarrow t_1(t'_1, \ldots, t'_n)]$$

and

$$s_2 = t[d \leftarrow t_2(t'_1, \ldots, t'_n)]$$

for some sequence of trees $t'_1, \ldots, t'_n \in \mathbf{T}_\Sigma(\mathbf{X}_n)$. In other words, the subtree $t_1(t'_1, \ldots, t'_n)$ of the tree s_1 is rewritten (or replaced) with the tree $t_2(t'_1, \ldots, t'_n)$ using the rewrite rule $t_1 \to t_2$. When the context of $\mathbf{P}$ is clearly known, $\Rightarrow_{\mathbf{P}}$ will simply be denoted $\Rightarrow$. Furthermore, let $\Rightarrow^+$ and $\Rightarrow^*$ denote the transitive and transitive reflexive closures of the relation $\Rightarrow$.

Rewrite rules are presented as general tree operators (in contrast with the section on context-free tree grammars) since trees will also be used to describe instantaneous descriptions (or configurations) of a tree pushdown automaton, and the actions of the automaton on these instantaneous descriptions will be presented using rewrite rules to modify the instantaneous descriptions. One should note that using rewrite rules to describe the actions of the tree pushdown automaton originates from Guessarian [22].

III. Context-free Tree Languages

3.1. *Context-free Tree Grammars*

Definition 3.1.1. A *context-free tree grammar* (CFTG for short) G is a quadruple $(\Phi, \Sigma, \mathbf{P}, S)$, where

Φ is a finite ranked alphabet of *nonterminal symbols*,

Σ is a finite ranked alphabet of *terminal symbols*,

$\mathbf{P}$ is a finite set of rewrite rules in $\mathbf{T}_\Phi(\mathbf{X}_m) \times \mathbf{T}_{\Phi \cup \Sigma}(\mathbf{X}_m)$ called *productions*, where $m = \max\{r(\gamma) \mid \gamma \in \Phi\}$ and each rewrite rule is of the form $\gamma(x_1, \ldots, x_{r(\gamma)}) \to t$ such that $t \in \mathbf{T}_{\Phi \cup \Sigma}(\mathbf{X}_{r(\gamma)})$, and

$\mathbf{S}$ is a designated symbol of Φ of rank zero called the *start symbol.*

For notational convenience, trees of the form $\gamma(x_1, \ldots, x_{r(\gamma)})$ will be denoted in vector form as $\gamma(\mathbf{x})$. In general, upper case letters such as $\mathbf{F}$, $\mathbf{G}$, $\mathbf{H}$,..., will be used to denote nonterminal symbols while lower case letters such as $\mathbf{f}$, $\mathbf{g}$, $\mathbf{h}$,..., will be used to denote terminal function symbols, and lower case letters such as $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$,..., will be

used to denote terminal constants. Depending on the context, G will also be used to denote a tree grammar. Furthermore, a sequence of rewrite steps will be called a *derivation.*

DEFINITION 3.1.2. The *language* generated by a CFTG $G=(\Phi, \Sigma, \mathbf{P}, S)$, denoted $L(G)$, is the set of trees $L(G)=\{t \in \mathbf{T}_\Sigma \mid S \Rightarrow_P^* t\}$. Furthermore, a *context-free tree language* (CFTL) is any tree language generated by a CFTG.

One should note that unlike string grammars, derivations are not commutative in the sense that if $t_1 \Rightarrow t_2$ using a production $F(\mathbf{x}) \rightarrow s_1$, and $t_2 \Rightarrow t_3$ using a production $G(\mathbf{x}) \rightarrow s_2$, it is not necessarily the case that there exists a tree t_2' such that $t_1 \Rightarrow t_2'$ using $G(\mathbf{x}) \rightarrow s_2$ and $t_2' \Rightarrow t_3$ using $F(\mathbf{x}) \rightarrow s_1$. This result is well known and has been shown by Engelfriet and Schmidt [12] and Nivat [26]. Furthermore, Engelfriet and Schmidt [12] (or Nivat [26]) have shown that the set of terminal trees derivable from a CFTG is equivalent to the set of terminal trees derivable from the CFTG using an OI (outside-in) derivation (i.e., derivations in which only topmost nonterminals are rewritten).

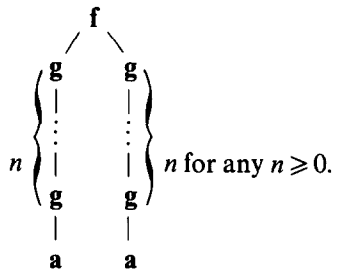
DEFINITION 3.1.3. Given a CFTG $G=(\Phi, \Sigma, \mathbf{P}, S)$ and two trees $t_1, t_2 \in \mathbf{T}_{\Phi \cup \Sigma}$, t_1 *rewrites to* t_2 under an OI derivation, denoted $t_1 \Rightarrow_{\mathbf{P}}^{OI} t_2$, iff $t_1 \Rightarrow_{\mathbf{P}} t_2$ such that $t_1 = s[d \leftarrow F(t_1', \ldots, t_{r(F)}')]$, $t_2 = s[d \leftarrow t(s_1', \ldots, t_{r(F)}')]$, $F(\mathbf{x}) \rightarrow t \in \mathbf{P}$, and for all prefixes d' of d, when $d' \neq d$, $t(d') \notin \Phi$ (*Note.* $\notin$ denotes set exclusion).

In the remainder of this paper, it will be assumed that all rewrite steps are OI unless otherwise specified. Hence, $\Rightarrow_{\mathbf{P}}^{OI}$ will simply be denoted as $\Rightarrow$ unless the context of $\mathbf{P}$ is not explicitly known. In the latter case, $\Rightarrow_{\mathbf{P}}^{OI}$ will be denoted as $\Rightarrow_{\mathbf{P}}$.

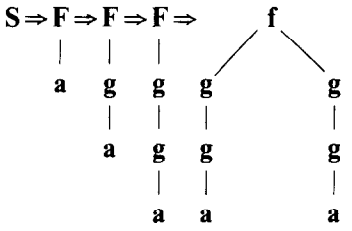
EXAMPLE 3.1.1. Let $G_1=(\Phi, \Sigma, \mathbf{P}, S)$ be a CFTG, where $\Sigma=\{f, g, a\}$ with $r(f)=2$, $r(g)=1$, and $r(a)=0$; $\Phi=\{\mathbf{S}, \mathbf{F}\}$, where $r(\mathbf{S})=0$ and $r(\mathbf{F})=1$; and

$$\mathbf{P}=\{\mathbf{S}\rightarrow \mathbf{F}(\mathbf{a}, \mathbf{x}),\ \mathbf{F}\rightarrow \mathbf{F}(\mathbf{g}(\mathbf{x}), \mathbf{x}),\ \mathbf{F}\rightarrow \mathbf{f}(\mathbf{x}, \mathbf{x})\}.$$

$L(G_1)$ is the set of trees of the form



A sample derivation is as follows:



3.2. *Chomsky Normal Form*

In a CFTG, there is no "a priori" bound on the size of the right-hand sides of productions. Proofs can be simplified if the right-hand side of a production has the property that the number of terminal and nonterminal symbols occurring in the tree, is bounded by two (as in Chomsky normal form for string grammars [4]). Schimpf [29] has shown that one can convert any CFTG into a corresponding definition of Chomsky normal form for trees.

DEFINITION 3.2.1. A CFTG $G=(\Phi, \Sigma, \mathbf{P}, S)$ is said to be in *Chomsky normal form* (CNF) iff every production $p \in \mathbf{P}$ is in one of the following three forms:

(1) $p=F(\mathbf{x})\to t$, where for all $d\in(\mathrm{dom}(t)-\mathrm{var}(t))$, $t(d)\in\Phi$, and $|\mathrm{dom}(t)-\mathrm{var}(t)|=2$ (i.e., contains two nonterminal symbols and no terminal symbols).

(2) $p=F(\mathbf{x})\to t$, where for all $d\in(\mathrm{dom}(t)-\mathrm{var}(t))$, $t(d)\in\Sigma$, and $|\mathrm{dom}(t)-\mathrm{var}(t)|=1$ (i.e., contains one terminal and no nonterminal symbols).

(3) $p=F(\mathbf{x})\to x$, where $x\in\mathbf{X}_{r(F)}$ (i.e., is an epsilon rule).

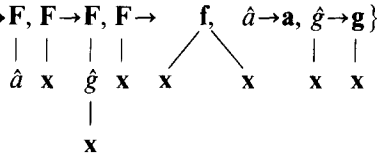
The following theorem from Schimpf [29] (which was proved using methods similar to those used in the string case) is given without proof.

THEOREM 3.2.1. *Given a CFTG $G=(\Phi, \Sigma, \mathbf{P}, S)$, there exists a CFTG $G'=(\Phi', \Sigma', \mathbf{P}', S')$ such that $L(G)=L(G')$ and G' is in CNF.*

COROLLARY. *Let C_{TG} denote the class of languages generated by CFTGs and C_{CNF} denote the class of languages generated by CFTGs in CNF. Clearly $C_{\mathrm{TG}}=C_{\mathrm{CNF}}$.*

EXAMPLE 3.2.1. Let $G_1=(\Phi, \Sigma, \mathbf{P}, S)$ and $G_2=(\Phi', \Sigma, \mathbf{P}', S)$ be CFTGs, where G_1 is defined as in Example 3.1.1, $\Phi'=\{\mathbf{S}, \mathbf{F}, \hat{a}, \hat{g}\}$, where $r(\mathbf{S})=r(\hat{a})=0$ and $r(\mathbf{F})=r(\hat{g})=1$, and

$$\mathbf{P}'=\{\mathbf{S}\to\mathbf{F},\ \mathbf{F}\to\mathbf{F},\ \mathbf{F}\to\ \mathbf{f},\ \hat{a}\to\mathbf{a},\ \hat{g}\to\mathbf{g}\}.$$



Clearly G_2 is in CNF and $L(G_1)=L(G_2)$.

IV. Tree Pushdown Automata

A *tree pushdown automaton* (TPDA) operates in the same manner as a standard bottom-up tree automaton [3, 8, 32, 31] except that there is an internal memory consisting of a finite sequence of trees (called tree stacks), where there is exactly one tree stack for each read-head in the bottom-up tree automaton. Furthermore, TPDAs correspond to standard (string) pushdown automata [23] in the same manner that bottom-up tree automata correspond to finite (string) automata [23]. Each tree in the internal memory of the TPDA is called a tree stack since each tree stack is treated like a stack in the sense that a TPDA can only read the root of each tree stack, and nodes can only be added (pushed) or deleted (popped) at the root of each tree stack.

Definition 4.1. A *tree pushdown automaton* (TPDA) is a sextuple $(\Sigma, \Gamma, \Delta, \square, \bot, \pm)$;

Σ is a finite ranked alphabet of *input symbols*,

Γ is a finite ranked alphabet of *stack symbols*,

$\square \notin \Sigma \cup \Gamma$ is a reserved symbol of rank 2 denoting a read-head,

$\pm \notin \Sigma \cup \Gamma$ is a reserved constant denoting the *initial tree stack*,

$\bot \in \Gamma$ is a reserved constant denoting the *empty tree, and*

$\Delta: \Sigma \cup \mathbf{T}_\Gamma(\mathbf{X}_m) \to 2^\Gamma$ is a partial function called the *transition function*, where $m = \max\{r(\gamma) \mid \gamma \in \Gamma\}$ and every transition is in one of the following two forms:

Shift-moves: $\gamma \in \Delta(\sigma)$, where $\sigma \in \Sigma$ and $r(\sigma) = r(\gamma)$,

Reduce-moves: $\gamma \in \Delta(t)$, where $r(\gamma) = m$ and $t \in \mathbf{T}_\Gamma(\mathbf{X}_m)$.

To provide a "snapshot" description of a TPDA, a single tree will be used. Such a tree contains the input tree, the read-heads (where the position of the read-heads is implied by their relative position in the tree), and the tree stack associated with each read-head. Informally, the format of this tree which describes the TPDA is as follows: The read-heads (denoted by the special symbol $\square$) are inserted into the input tree such that the read-heads separate the scanned and unscanned portions of the input tree. The scanned portions of the input are nodes occurring in the leftmost subtrees of nodes labeled by a read-head while the unscanned portion of the input tree are nodes that are ancestors of nodes labeled by a read-head. Furthermore, the rightmost subtree of a node labeled by a read-head is the tree stack associated with that read-head.

Definition 4.2. An *instantaneous description* (ID) of a TPDA $M = (\Sigma, \Gamma, \Delta, \square, \bot, \pm)$ is a tree $t \in \mathbf{T}_{\Sigma \cup \Gamma \cup \{\square, \pm\}}$, where

(i) For all $d \in \text{leaf}(t)$, there exists a prefix d' of d such that $t(d') = \square$;

(ii) If $t(d) = \square$ then for all prefixes d' of d such that $d' \neq d$, $t(d') \in \Sigma$;

(iii) If $t(d) = \square$ then for all $d1d' \in \text{dom}(t)$, $t(d1d') \in \Sigma$;

(iv) If $t(d) = \square$ then for all $d2d' \in \text{dom}(t)$, $t(d2d') \in \Gamma \cup \{\pm\}$.

Note that conditions (i) and (ii) guarantee that the read-heads "slice" the input tree while condition (iv) guarantees that the rightmost subtree of a read-head is a tree stack. Hence, for the input tree $t' = t(t_1,..., t_m)$, where each x_i $(1 \leqslant i \leqslant m)$ occurs exactly once in the tree t, the instantaneous description $t(\square(t_1, t_1'),..., \square(t_m, t_m'))$ states that the scanned portion of the input tree associated with the ith read-head is t_i and its corresponding tree stack is t_i' while the tree t represents the unscanned portion of the input tree. Furthermore, the *initial instantaneous description* for the input tree t', denoted $id_0(t')$, is the tree defined by the set of ordered pairs

$$\{(d, \sigma) \mid (d, \sigma) \in t', r(\sigma) \neq 0\} \cup \{(d, \square), (d \cdot 1, \sigma), (d \cdot 2, \pm) \mid (d, \sigma) \in t', r(\sigma) = 0\}.$$

The above representation of instantaneous descriptions has been purposely chosen to allow us to express the transition rules of a TPDA using rewrite rules instead of having to present a new relation to describe how instantaneous descriptions are updated (the notion of using rewrite rules to express transition rules of the TPDA originates from Guessarian [27]).

DEFINITION 4.3. Given a TPDA $D = (\Sigma, \Gamma, \Delta, \square, \perp, \pm)$, the set of *rewrite rules describing the transition function* Δ, denoted Δ', is defined as follows:

(i) for all $\gamma \in \Delta(\sigma)$, where $\sigma \in \Sigma$ and $r(\sigma) = r(\gamma) = 0$, $\square(\sigma, \pm) \to \square(\sigma, \gamma) \in \Delta'$
(*Note.* A rewrite rule of this form describes a shift (or read) move of the leaf σ);

(ii) for all $\gamma \in \Delta(\sigma)$, where $\sigma \in \Sigma$ and $r(\sigma) = r(\gamma) = m \geqslant 1$, $\sigma(\square(x_1, x_{m+1}),..., \square(x_m, x_{2m})) \to \square(\sigma(x_1,..., x_m), \gamma(x_{m+1},..., x_{2m})) \in \Delta'$
(*Note.* Each variable x_i $(1 \leqslant i \leqslant m)$ is used to describe how the scanned portion of the input tree, associated with the ith read-head, will be manipulated while x_{m+i} shows how the corresponding tree stack will be manipulated. Hence, a rewrite rule of this form describes a shift move over a node labeled with σ, where the m read-heads immediately below the node labeled with σ are merged into a single read-head and the corresponding m tree stacks are merged together using the stack symbol γ);

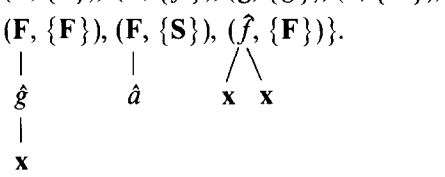
(iii) for all $\gamma \in \Delta(t)$, where $r(\gamma) = m$ and $t \in \mathbf{T}_\Gamma(\mathbf{X}_m)$, $\square(x_{m+1}, t) \to \square(x_{m+1}, \gamma(x_1,..., x_m)) \in \Delta'$
(*Note.* A rewrite rule of this form describes a reduce move (tree stack update move). The variable x_{m+1} is used to describe how the scanned input tree, associated with the read-head being reduced, is manipulated. Hence a reduce move does not advance any read-heads but removes (or pops) the supertree t from the tree stack and unites the orphaned subtrees with the new parent γ. Since the set of rewrite rules Δ' (not the transition function Δ) is used to describe how instantaneous descriptions are updated, the remainder of this paper will assume that subsequent references to Δ are actually referencing the corresponding set of rewrite rules Δ).

DEFINITION 4.4. The *language* accepted by a TPDA $M=(\Sigma, \Gamma, \Delta, \square, \perp, \pm)$, denoted $T(M)$, is the set $\{t \in T_\Sigma \mid id_0(t) \Rightarrow_\Delta^* \square(t, \perp)\}$. Furthermore, each rewrite performed is called a *computation.*

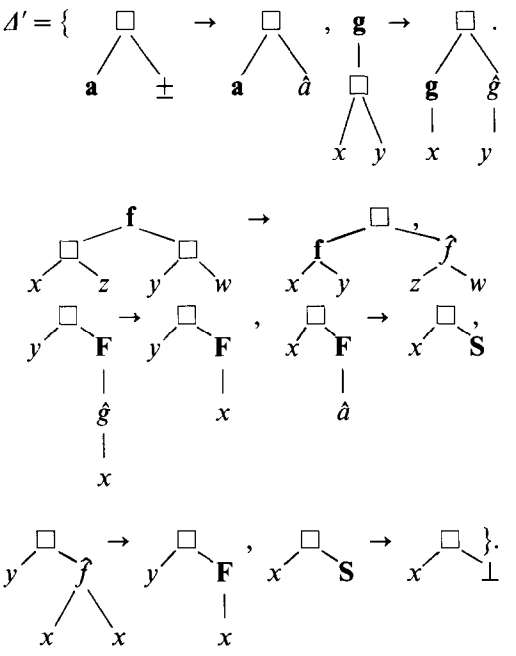
In other words, acceptance occurs as the read-heads can be advanced to the root of the tree and the corresponding tree stack is empty.

EXAMPLE 4.1. Consider the language generated by the CFTG G in Example 3.1. Let $M=(\Sigma, \Gamma, \Delta, \square, \perp, \pm)$ be a TPDA, where $\Sigma=\{a, f, g\}$, where $r(a)=0$, $r(f)=2$, and $r(g)=1$; $\Gamma=\{\hat{a}, \hat{f}, \hat{g}, F, S, \perp, \pm\}$, where $r(a)=r(S)=r(\perp)=r(\pm)=0$, $r(\hat{f})=2$, and $r(\hat{g})=r(F)=1$; and

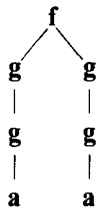
$$\Delta=\{(\mathbf{a}, \{\hat{a}\}), (\mathbf{f}, \{\hat{f}\}), (\mathbf{g}, \{\hat{g}\}), (\mathbf{S}, \{\perp\}),$$
$$(\mathbf{F}, \{\mathbf{F}\}), (\mathbf{F}, \{\mathbf{S}\}), (\hat{f}, \{\mathbf{F}\})\}.$$



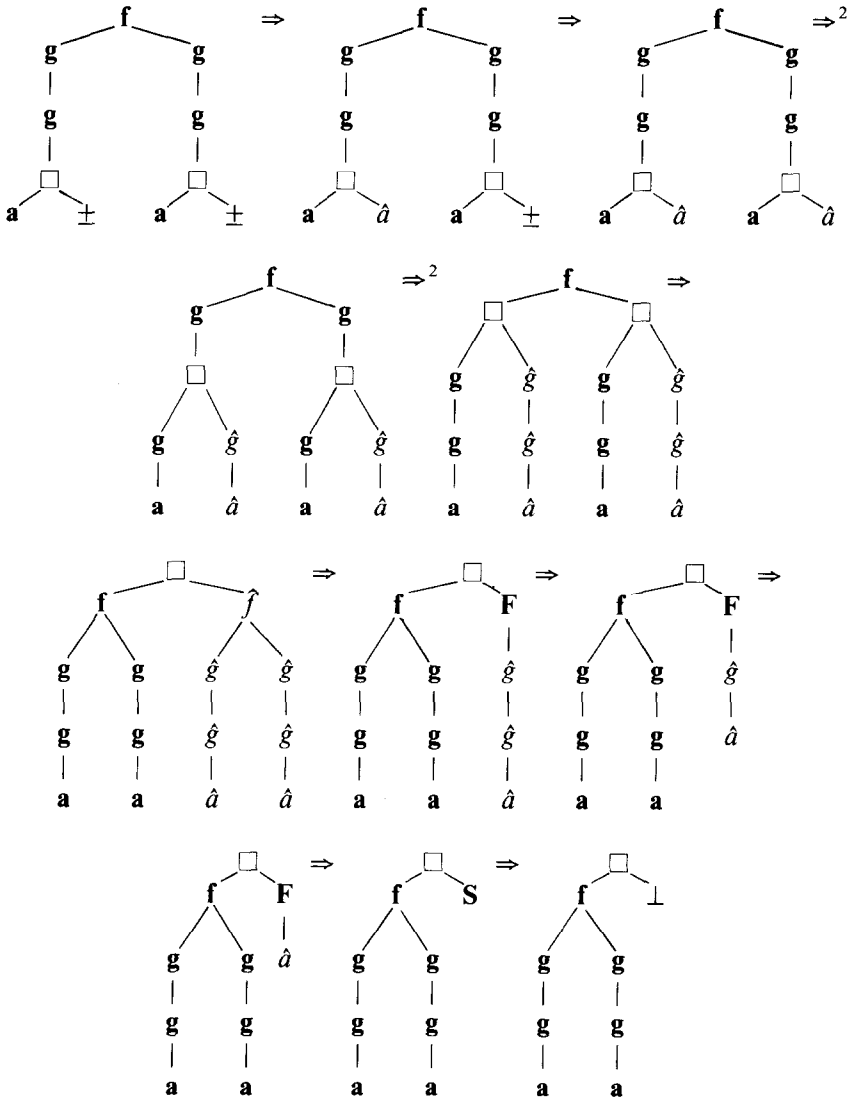
The set of rewrite rules describing the transition function Δ is the set



Clearly, $T(M) = L(G)$. For example, the tree



is accepted as follows:



V. Equivalence with CFTLs

This section shows that the class of CFTLs is identical to the class of languages accepted by TPDAs. This result is true by proving that every CFTG in CNF can be converted to a TPDA and conversely.

5.1. *Converting CFTGs to TPDAs*

The method for converting a CFTG in CNF to a TPDA resembles the conversion used to convert a context-free string grammar in CNF to a pushdown automaton where the moves of the pushdown automaton simulate legal derivations (for the string case, see [23, 5]). However, due to the bottom-up nature of the TPDA, the method presented here simulates the inverse of derivations.

Definition 5.1.1. Given a CFTG $G=(\Phi, \Sigma, \mathbf{P}, S)$ in CNF, let the corresponding TPDA $M=(\Sigma, \Gamma, \Delta, \square, \perp, \pm)$, denoted $TA(G)$, be defined where $\Gamma=\Phi \cup \mathbf{P} \cup \{\perp\}$, where $r(\perp)=0$, and for each production $F(\mathbf{x}) \rightarrow t \in \mathbf{P}$, $r(F(\mathbf{x}) \rightarrow t)=r(t(\varepsilon))$, and Δ is defined by the following four conditions:

(i) $\square(x, S) \rightarrow \square(x, \perp) \in \Delta$.

(ii) If $p=F(\mathbf{x}) \rightarrow a \in \mathbf{P}$, where $a \in \Sigma$ and $r(a)=0$, then

(a) $\square(a, \pm) \rightarrow \square(a, p) \in \Delta$,

(b) $\square(a, p) \rightarrow \square(a, F(\mathbf{x})) \in \Delta$.

(iii) If $p=F(\mathbf{x}) \rightarrow f(x_1', \ldots, x_q') \in \mathbf{P}$, where $f \in \Sigma$, $r(f)=q>0$, $r(F)=m$, and for all i, $1 \leqslant i \leqslant q$, $x_i' \in \mathbf{X}_m$, then

(a) $f(\square(x_1, x_{q+1}), \ldots, \square(x_q, x_{2q})) \rightarrow \square(f(x_1, \ldots, x_q), p(x_{q+1}, \ldots, x_{2q})) \in \Delta$,

(b) $\square(x_{m+1}, p(x_1', \ldots, x_q')) \rightarrow \square(x_{m+1}, F(\mathbf{x})) \in \Delta$.

(iv) If $p=F(\mathbf{x}) \rightarrow t$, where $r(F)=m$ and $t \in \mathbf{T}_{\Phi}(\mathbf{X}_m)$, then $\square(x_{m+1}, t) \rightarrow \square(x_{m+1}, F(\mathbf{x})) \in \Delta$.

Note that condition (i) causes acceptance if the tree is derivable from the start symbol, condition (ii) simulates the derivation $F(t_1, \ldots, t_m) \Rightarrow a$, condition (iii) simulates the derivation $F(t_1, \ldots, t_m) \Rightarrow f(t_1', \ldots, t_m')$, where $t_i'=t_j$ iff $x_1'=x_j$, and condition (iv) simulates the derivation $F(t_1, \ldots, t_m) \Rightarrow t(t_1, \ldots, t_m)$. Furthermore, conditions (ii) and (iii) are performed in two steps. The reason for the two steps is that a production may rearrange, duplicate, or erase subtrees depending on the occurrences of variables on the right-hand side of the production. However, a shift-move does not allow such reconfigurations. Hence, a second step (the reduce-move) was added to handle these reconfigurations.

Lemma 5.1.1. *Given a CFTG* $G=(\Phi, \Sigma, \mathbf{P}, S)$ *in CNF, the TPDA* $TA(G)=(\Sigma, \Gamma, \Delta, \square, \perp, \pm)$, *any* $F \in \Phi$, *where* $r(F)=m$, *any sequence of trees* $t_1, \ldots, t_m \in \mathbf{T}_{\Phi}$, *any* $t \in \mathbf{T}_{\Sigma}$, *and any* $n \geqslant 1$:

(1) *If* $F(t_1,..., t_m) \Rightarrow_{\mathbf{P}}^{n} t$ *then* $id_0(t) \Rightarrow_{\Delta}^{*} \square(t, F(t_1,..., t_m))$.

(2) *If* $id_0(t) \Rightarrow_{\Delta}^{n} \square(t, F(t_1,..., t_m))$ *then* $F(t_1,..., s_n) \Rightarrow_{\mathbf{P}}^{*} t$.

Proof 1. **Base case.** $F(t_1,..., t_m) \Rightarrow_{\mathbf{P}} a$ using production $p = F(\mathbf{x}) \rightarrow a$, where $a \in \Sigma$ and $r(a) = 0$. Clearly $id_0(a) = \square(a, \pm) \Rightarrow_{\Delta} \square(a, p) \Rightarrow_{\Delta} \square(a, F(t_1,..., t_m))$.

Inductive step. $F(t_1,..., t_m) \Rightarrow_{\mathbf{P}} t'(t_1,..., t_m) \Rightarrow_{\mathbf{P}}^{n} t$ using production $p = F(\mathbf{x}) \rightarrow t'$, where $n > 0$.

Case 1. $t' \in \mathbf{T}_{\Phi}(\mathbf{X}_m)$. Clearly $id_0(t) \Rightarrow_{\Delta}^{*} \square(t, t'(t_1,..., t_m)) \Rightarrow_{\Delta} \square(t, F(t_1,..., t_m))$.

Case 2. $t' = f(x_1',..., x_q')$, where $f \in \Sigma$, $r(f) = q > 0$, and for all i, $1 \leqslant i \leqslant q$, $x_i' \in \mathbf{X}_m$. Hence $t'(t_1,..., t_q) = f(t_1',..., t_q')$ and $t = f(t/1,..., t/q)$, where for all i, $1 \leqslant i \leqslant q$, $t_i' = t_j$ iff $x_i' = x_j$. Therefore $id_0(t) \Rightarrow_{\Delta}^{*} f(\square(t/1, t_1'), id_0(t/2),..., id_0(t/q)) \Rightarrow_{\Delta}^{*} \cdots \Rightarrow_{\Delta}^{*} f(\square(t/1, t_1'),..., \square(t/q, t_q')) \Rightarrow_{\Delta} \square(t, p(t_1',..., t_q')) \Rightarrow_{\Delta} \square(t, F(t_1,..., t_q))$.

Proof 2. **Base case.** $\square(a, \pm) \Rightarrow_{\Delta} \square(a, p) \Rightarrow_{\Delta} \square(a, F(t_1,..., t_m))$, where $a \in \Sigma$ and $r(a) = 0$. Clearly $F(t_1,..., t_m) \Rightarrow_{\mathbf{P}} a$.

Inductive step. $id_0(t) \Rightarrow_{\Delta}^{n} id \Rightarrow_{\Delta} \square(t, F(t_1,..., t_m))$, where $n > 2$ and id is some instantaneous description.

Case 1. $id = \square(t, t'(t_1,..., t_m))$, where $t' \in \mathbf{T}_{\Phi}(\mathbf{X}_m)$. Hence $F(t_1,..., t_m) \Rightarrow_{\mathbf{P}} t'(t_1,..., t_m) \Rightarrow_{\mathbf{P}}^{*} t$.

Case 2. $id = \square(t, p(t_1',..., t_q'))$, where $t' = f(x_1',..., x_q')$, $f \in \Sigma$, $r(f) = q > 0$, and for all i, $1 \leqslant i \leqslant q$, $x_i' \in \mathbf{X}_m$ and if $x_i' = x_j$ then $t_i' = t_j$. By inspection of the definition of $TA(G)$, id is computable iff $id_0(t) \Rightarrow_{\Delta}^{n-1} id'$, where $id' \Rightarrow_{\Delta} id$ using the transition $f(\square(x_1, x_{q+1}),..., \square(x_q, x_{2q})) \rightarrow \square(f(x_1,..., x_q), p(x_{q+1},..., x_{2q}))$. But then it must be the case that $id' = f(\square(t/1, t_1'),..., \square(t/q, t_q'))$. Therefore $F(t_1,..., t_m) \Rightarrow_{\mathbf{P}} t'(t_1,..., t_m) \Rightarrow_{\mathbf{P}}^{*} t$.

THEOREM 5.1.1. *Given any CFTG* $G = (\Phi, \Sigma, \mathbf{P}, S)$ *in CNF and the TPDA* $TA(G) = M = (\Sigma, \Gamma, \Delta, \square, \perp, \pm)$, $L(G) = T(M)$.

Proof. By definition $L(G) = \{t \mid S \Rightarrow_{\mathbf{P}}^{*} t, t \in \mathbf{T}_{\Sigma}\}$ and $T(M) = \{t \mid t \in \mathbf{T}_{\Sigma}, id_0(t) \Rightarrow_{\Delta}^{*} \square(t, \perp)\}$. Let $t \in \mathbf{T}_{\Sigma}$ be any tree such that $S \Rightarrow_{\mathbf{P}}^{*} t$. By Lemma 5.1.1, $id_0(t) \Rightarrow_{\Delta}^{*} \square(t, S)$. By the definition of $TA(G)$, $\square(t, S) \Rightarrow_{\Delta} \square(t, \perp)$. Hence $t \in T(M)$ and $L(G) \subseteq T(M)$. On the other hand, let $t \in \mathbf{T}_{\Sigma}$ be an tree such that $id_0(t) \Rightarrow_{\Delta}^{*} \square(t, \perp)$. By inspection of the definition of $TA(G)$, it must be the case that $id_0(t) \Rightarrow_{\Delta}^{*} \square(t, S) \Rightarrow_{\Delta} \square(t, \perp)$. By Lemma 5.1.1, $S \Rightarrow_{\mathbf{P}}^{*} t$. Hence $t \in L(G)$ and $T(M) \subseteq L(G)$. Therefore $L(G) = T(M)$.

5.2. *Converting TPDAs to CFTGs*

The method for converting a TPDA to a CFTG is straightforward and does not need to mimic the conversions typically used to convert a PDA to a context-free string grammar [22]. The simplification which allows this straightforward conversion is that the definition of a TPDA presented here does not have explicit states

nor does it have stack lookback (i.e., reference to top of tree stack when shifting, or the use of the current top of stack after popping the right-hand side of a production off the stack while performing a reduce-move). Hence, instead of having to create a new set of nonterminals to capture changes in state, tree stack, and tree stack lookback, it is only necessary to capture changes in tree stack (i.e., the set of nonterminals of the created CFTG is simply the tree stack alphabet). Further, it can be shown that these added conditions do not increase the size of class of languages accepted by such automata (see Schimpf [29] for details of the proofs).

DEFINITION 5.2.1. Given a TPDA $M=(\Sigma, \Gamma, \Delta, \square, \perp, \pm)$, let the corresponding CFTG $(\Gamma, \Sigma, \mathbf{P}, \perp)$, denoted $TG(M)$, be defined such that $\mathbf{P}$ is defined by the following three conditions:

(i) If $\square(\sigma, \pm)\to\square(\sigma, \gamma)\in\Delta$ is a shift-move, where $\sigma\in\Sigma$, $\gamma\in\Gamma$, and $r(\sigma)=r(\gamma)=0$, then $\gamma\to\sigma\in\mathbf{P}$.

(ii) If $\sigma(\square(x_1, x_{q+1}),..., \square(x_q, x_{2q}))\to\square(\sigma(x_1,..., x_q), \gamma(x_{q+1},..., x_{2q}))\in\Delta$ is a shift-move, where $\sigma\in\Sigma$, $\gamma\in\Gamma$, and $r(\sigma)=r(\gamma)=q$, then $\gamma(\mathbf{x})\to\sigma(\mathbf{x})\in\mathbf{P}$.

(iii) If $\square(x_{q+1}, t)\to\square(x_{q+1}, \gamma(\mathbf{x}))\in\Delta$ is a reduce-move, where $\gamma\in\Gamma$, $r(\gamma)=q$, and $t\in\mathbf{T}_\Gamma(\mathbf{X}_q)$, then $\gamma(\mathbf{x})\to t\in\mathbf{P}$.

LEMMA 5.2.1. *Given a TDPA* $M=(\Sigma, \Gamma, \Delta, \square, \perp, \pm)$, *its corresponding CFTG* $TG(M)=(\Gamma, \Sigma, \mathbf{P}, S)$, *any* $\gamma\in\Gamma$, *where* $r(\gamma)=m$, *any sequence of trees* $t_1,..., t_m\in\mathbf{T}_\Gamma$, *any* $t\in\mathbf{T}_\Sigma$, *and any* $n>0$,

(1) *if* $id_0(t)\Rightarrow_\Delta^n\square(t, F(t_1,..., t_n))$ *then* $F(t_1,..., t_m)\Rightarrow_\mathbf{P}^* t$

(2) *if* $F(t_1,..., t_m)\Rightarrow_\mathbf{P}^n t$ *then* $id_0(t)\Rightarrow_\Delta^* \square(t, F(t_1,..., t_m))$.

Proof 1. **Base case.** $id_0(t)\Rightarrow_\Delta\square(a, F(t_1,..., t_m))$. Clearly $t(\varepsilon)=a\in\Sigma$, where $r(a)=r(F)=0$. Hence $F(\mathbf{x})\to a\in\mathbf{P}$ and $F(t_1,..., t_m)\Rightarrow_\mathbf{P} a$.

Inductive case. $id_0(t)\Rightarrow_\Delta^{n-1} id\Rightarrow_\Delta\square(t, F(t_1,..., t_m))$, where id is an instantaneous description.

Case 1 (shift-move). $id=f(\square(t/1, t_1),..., \square(t/m, t_m))$, where $r(f)=r(F)=m$. Hence $F(t_1,..., t_m)\Rightarrow_\mathbf{P} f(t_1,..., t_m)\Rightarrow_\mathbf{P}^* f(t/1, t_2,..., t_m)\Rightarrow_\mathbf{P}^*\cdots\Rightarrow_\mathbf{P}^* f(t/1,..., t/m)=t$.

Case 2 (reduce-move). $id=\square(t, t'(t_1,..., t_m))$, where $\square(x_{m+1}, t')\to\square(x_{m+1}, F(\mathbf{x}))\in\Delta$. Hence $F(t_1,..., t_m)\Rightarrow_\mathbf{P} t'(t_1,..., t_m)\Rightarrow_\mathbf{P}^* t$.

Proof 2. **Base case.** $F(t_1,..., t_m)\Rightarrow_\mathbf{P} t$. By definition, it must be the case that $\square(a, \perp)\to\square(a, F)\in\Delta$, where $t(\varepsilon)=a\in\Sigma$, $F\in\Gamma$, and $r(a)=t(F)=0$. Hence $id_0(a)\Rightarrow_\Delta\square(a, F)$.

Inductive step. $F(t_1,..., t_m)\Rightarrow_\mathbf{P} t'(t_1,..., t_m)\Rightarrow_\mathbf{P}^{n-1} t$.

Case 1 (shift-move). $t'=f(\mathbf{x})$, where $f\in\Sigma$ and $r(f)=m$. Hence $id_0(t)\Rightarrow_\Delta^* f(\square(t/1, t_1), id_0(t/2),..., id_0(t/m))\Rightarrow_\Delta^*\cdots\Rightarrow_\Delta^* f(\square(t/1, t_1),..., \square(t/m, t_m))\Rightarrow_\Delta\square(t, F(t_1,..., t_m))$.

Case 2 (reduce-move). $t' \in \mathbf{T}_\Gamma(\mathbf{X}_m)$. Clearly $id_0(t) \Rightarrow_\Delta^* \Box(t, t'(t_1,..., t_m)) \Rightarrow_\Delta \Box(t, F(t_1,..., t_m))$.

THEOREM 5.2.1. *Given any TPDA* $M = (\Sigma, \Gamma, \Delta, \Box, \perp, \pm)$ *and the corresponding CFTG* $TG(M) = G = (\Gamma, \Sigma, \mathbf{P}, \perp)$, $L(G) = T(M)$.

Proof. By definition $L(G) = \{t \mid \perp \Rightarrow_\mathbf{P}^* t, t \in \mathbf{T}_\Sigma\}$ and $T(M) = \{t \mid id_0(t) \Rightarrow_\Delta^* \Box(t, \perp), t \in \mathbf{T}_\Sigma\}$. Let $t \in \mathbf{T}_\Sigma$ be any tree such that $\perp \Rightarrow_\mathbf{P}^* t$. By Lemma 5.2.1 $id_0(t) \Rightarrow_\Delta^* \Box(t, \perp)$. Hence $t \in T(M)$ and $L(G) \subseteq T(M)$. On the other hand, let $t \in \mathbf{T}_\Sigma$ be any tree such that $id_0(t) \Rightarrow_\Delta^* \Box(t, \perp)$. By Lemma 5.2.1, $\perp \Rightarrow_\mathbf{P}^* t$. Hence $t \in L(G)$ and $T(M) \subseteq L(G)$. Therefore $L(G) = T(M)$.

5.3. *Equivalence*

Let C_{TG}, C_{CNF}, and C_{TPDA} denote the class of CFTLs, the class of CFTLs generated by CFTGs in CNF, and the class of tree languages accepted by TPDA, respectively.

THEOREM 5.3.1. $C_{\text{TG}} = C_{\text{CNF}} = C_{\text{TPDA}}$.

Proof. Directly follows from Theorems 3.2.1, 5.1.1, and 5.2.1, which clearly shows that

$$C_{\text{TG}} = C_{\text{CNF}} \subseteq C_{\text{TPDA}} \subseteq C_{\text{TG}}.$$

VI. CONCLUSION

This paper has presented a new form of automata called tree pushdown automata and showed that the power of tree pushdown automata is identical to that of context-free tree grammars. The methods shown here resemble the methods used to show the equivalence between context-free string grammars and pushdown automata, and generalize these results up to trees in a manner similar to the way a finite state automaton is generalized to a bottom-up tree automaton.

This generalization of results about pushdown automata and context-free string grammars up to trees extends far beyond what has been presented here. For example, besides showing the results of this paper, Schimpf [29] has also shown that the notion of *LR*-parsers and *LR*-parser generators [24] also lifts up to context-free tree grammars and is the object of a forthcoming paper. Another example is the top-down tree pushdown automaton presented by Guessarian [22] which presents similar results except that she has lifted top-down parsing methods instead of bottom-up. Hence, it appears that the results and parsing methodologies for the context-free string languages, can be generalized to tree parsing techniques. Furthermore, these results can then be applied to the yield of context-free tree languages (see Schimpf [29]) and result in even more powerful machinery which has the power to parse the indexed (or macro) languages (see [13, 14, 25]), as well as

produce a deterministic parser for a subset of the indexed languages (which is a superset of the deterministic context-free string languages and will be the object of a second forthcoming paper).

REFERENCES

1. A. V. AHO, Nested stack automata, *J. Assoc. Comput. Mach.* **16**, No. 3 (1969).
2. A. V. AHO AND J. D. ULLMAN, The theory of parsing, translation, and compiling, *Internat. J. Comput. Math.* **3** (1972).
3. J. R. BUCHI AND J. B. WRIGHT, Mathematical theory of automata, Notes on material presented by Buchi and Wright, Communications Sciences No. 403, Univ. of Michigan, Ann Arbor, Mich., 1960.
4. N. CHOMSKY, On certain formal properties of grammars, *Inform. and Control* **2** (1959).
5. N. CHOMSKY, "Context-free Grammars and Pushdown Storage," MIT Research Lab of Electronics Quarterly Progress Report No. 65, 1962.
6. B. COURCELLE, Completeness results for equivalence of recursive schemes, *J. Comput. System Sci.* **12**, No. 2 (1976).
7. B. COURCELLE, A representation of trees by languages, I, II, *Theoret. Comput. Sci.* **6**, **7** (1981).
8. J. E. DONER, Tree acceptors and some of their applications, *J. Comput. System Sci.* **4** (1970).
9. P. J. DOWNEY, "Formal Languages and Recursion Schemes," Ph.D. dissertation, Harvard University, Cambridge, Mass., 1974.
10. J. ENGELFRIET, Some open questions and recent results on tree transducers and tree languages, Memorandum No. 293, Technische Hogeschool Twente, Enschede, The Netherlands, 1980.
11. J. ENGELFRIET, Some open questions and recent results in tree transducers and tree languages, *in* "Formal Language Theory" (R. Book, Ed.), Academic Press, New York, 1980.
12. J. ENGELFRIET AND E. M. SCHMIDT, IO and IO I, II, *J. Comput. System Sci.* **15**, **16** (1977–1978).
13. M. J. FISCHER, "Grammars with Macro-like Productions," Doctoral dissertation, Harvard University, Cambridge, Mass., 1968.
14. M. J. FISCHER, Grammars with macro-like productions, *in* "9th Sympos., Switching and Automata Theory," 1969.
15. K. S. FU, "Syntactic Methods in Pattern Recognition," Academic Press, New York, 1974.
16. K. S. FU AND B. K. BHARGAVA, Tree systems for syntactic pattern recognition, *IEEE Trans. Comput.* **C22** (1973).
17. J. H. GALLIER, "Alternative Proofs and New Results about Recursion Schemes and DPDA's," Report No. MS-CIS-80-7, Dept. of CIS, Univ. of Pennsylvania, 1980.
18. J. H. GALLIER, DPDA's in "atomic normal form" and applications to equivalence problems, *Theoretical Computer Science* **14** (2) (1981), 155–196.
19. S. GORN, Processors for infinite codes of the Shannon–Fano type, *in* "Proceedings of the Symposium on Mathematical Theory of Automata," 1962.
20. S. GORN, "Explicit Definitions and Linguistic Dominoes, Systems and Computer Science," J. Hart and Satoru Takasu, Eds.), 1965.
21. I. GUESSARIAN, On pushdown tree automata, *in* "Proceedings of 6th CAAP, Genoa," Lecture Notes in Computer Science, Springer-Verlag, New York/Berlin, 1981.
22. I. GUESSARIAN, "Pushdown Tree Automaton," Technical Report No. 82-28, LITP, University Paris VII, 1982.
23. M. A. HARRISON, "Introduction to Formal Language Theory," Addison–Wesley Reading, Mass., 1978.
24. D. E. KNUTH, On the translation of languages from left to right, *Inform. and Control* **8** (1965).
25. K. MEHLHORN, Parsing macro grammars top down, *Inform. and Control* **40**, No. 1 (1979).
26. M. NIVAT, On the interpretation of recursic polyadic program schemes, *in* "Proc., Symposia Mathematica" Vol. 15, Academic Press, New York, 1975.

27. W. C. ROUNDS, Context-free grammars on trees, *in* "IEEE 10th Annual Symposium on Switching and Automata Theory," 1969.
28. W. C. ROUNDS, Mappings and grammars on trees, *J. Math. System Theory* **4**, No. 3 (1970).
29. K. M. SCHIMPF, "A Parsing Method for Context-free Tree Languages," Ph.D. dissertation, University of Pennsylvania, Philadelphia, Pa., 1982.
30. J. W. THATCHER, Characterizing derivation trees of context-free grammars through a generalization of finite automata theory, *J. Comput. System Sci.* **1** (1967).
31. J. W. THATCHER, Tree automata: an informal survey, *in* "Currents in the Theory of Computing" (A. V. Aho, Ed.), Prentice–Hall, Englewood Cliffs, N.J., 1973.
32. J. W. THATCHER AND J. B. WRIGHT, Generalized finite automata theory with an application to a decision problem of second order logic, *J. Math. Systems Theory* **2** (1968).