Math. Systems Theory 16, 237–263 (1983)

Mathematical Systems Theory

Pushdown Tree Automata

Irène Guessarian

LITP–CNRS–UER de Mathématiques, Université Paris 7–T. 55/56, 2, Pl. Jussieu–75251 PARIS Cedex 05, France

Abstract. We define topdown pushdown tree automata (PDTA's) which extend the usual string pushdown automata by allowing trees instead of strings in both the input and the stack. We prove that PDTA's recognize the class of context-free tree languages. (Quasi)realtime and deterministic PDTA's accept the classes of Greibach and deterministic tree languages, respectively. Finally, PDTA's are shown to be equivalent to restricted PDTA's, whose stack is linear: this both yields a more operational way of recognizing context-free tree languages and connects them with the class of indexed languages.

1. Introduction

Many structures in computer science can be represented by trees: derivation trees, syntax directed translations, search in files, etc.... There was thus developed, at first, a theory of recognizable tree languages, tree grammars and tree transducers [3, 11, 25, 31]. Tree language theory has since been helpful in a broad range of domains, e.g. decision problems in logics [24], formal language theory [26, 31] and program scheme theory [7, 8, 16, 18, 22]. In order to be applicable to such a wide range of problems, the tree language theory had to be extended to allow context free tree languages; it thus became possible to model program scheme theory (which is our motivation) and also more general language theory (e.g. Aho's indexed languages [1, 10, 13, 15, 25]—since an indexed language is the yield of a context free tree language). Now, any language theory usually presents three complementary aspects: grammatical, set-theoretical or algebraic, and automaton theoretic. For recognizable tree languages, all three aspects have been well studied, even for infinitary languages [23, 24], and comprehensive accounts can be found in the above given references [3, 11, 31, 32]. For context free languages, the

grammatical point of view was first studied in [10, 15, 25], the algebraic aspects were considered in [7, 13, 18, 22], and comprehensive and exhaustive surveys can be found in [7, 12, 27]. But the automaton theoretical aspect has not been studied at all; it is merely hinted at in [7, 25], but we can say that the prevailing point of view is grammatical and no attempt has been made to find a more operational way of looking at context free tree languages. This paper is one of the first steps in that direction; a more general approach, introducing abstract families of automata and relating them to AFL's can be found in [14].

We define pushdown tree automata (PDTA's) which extend usual string pushdown automata [6, 19] by allowing trees instead of strings in both the input and the pushdown.

The machine reads its input like a finite top down tree automaton [31, 32], while scanning the top (root) of the store (the root is the only stack symbol accessible at a given moment). We prove that PDTA's recognize context-free tree languages. A different model, called TPDA is investigated in [28, 29]: it stresses the parsing standpoint. Henceforth, TPDA's process trees in a bottom-up manner. The advantage of TPDA's is that a wide subclass of context-free tree languages can be recognized by deterministic TPDA's. The trade-off is that TPDA's are of course much more complicated than PDTA's. The latter stick more closely to the behaviour of grammars and their pushdown consists of a single tree; whereas the former's pushdown consists of arbitrary sets of trees whose roots must be simultaneously accessible and they consult appropriate shift-reduce tables. PDTA's thus provide a parallel LL-parsing method for context free tree languages, as opposed to the parallel LR-parsing of TPDA's.

We then improve vastly that parsing and make it more operational by restricting the pushdown and allowing only words instead of trees on it: i.e. we linearize the pushdown; this would be difficult to realize with TPDA's and is also one of the advantages of our model. We show that the corresponding parsers, called RPDTA's, still recognize the class of context-free tree languages: the intuition behind our construction is the classical idea of storing return addresses of recursive calls in the pushdown. The technical application of this idea to formal language and automata theory first appears in [16] whose construction inspired ours. Meanwhile, this automaton standpoint gives very easy connections with Rounds' creative grammars [25] and Aho's indexed languages [1, 12, 15]; we hope that this different method of interrelating known classes of languages will help in giving more insight to tree language theory.

The paper is organized as follows: section 2 is devoted to fixing the notations. In section 3 we introduce PDTA's and prove that the class of languages recognized by PDTA's is exactly the class of context-free tree languages. In section 4 the classes of languages accepted by (quasi)real-time and deterministic automata are shown to coincide with Greibach and deterministic tree languages. In section 5, we show how every context-free tree language can be accepted by a restricted PDTA (with a linear stack). We prove that some special PDTA's can be viewed as indexed grammars.

This paper is self-contained. No previous knowledge of context-free tree languages is required. All introduced notions are carefully defined, illustrated via examples and related to the literature. This makes the paper slightly longer than strictly needed and hopefully more readable. We hope the tree language experts

will forgive us this extra length: we tried to make it easy for them to hop their way throughout the paper.

2. Notations

Let F (resp. Φ) be a finite ranked alphabet of base function symbols (resp. of variable function symbols). The rank of a symbol s in $F \cup \Phi$ is denoted by $r(s)$. Symbols in F are denoted $f, g, h, \ldots$ if they have rank ≥ 1, and $a, b, \ldots$ if they have rank 0; F_i denotes the symbols of rank i in F. Symbols in Φ are denoted $G, H, \ldots$. Let V be a set of variables: the variables have rank 0 and are denoted by $u, v, w, \ldots$ possibly with indices.

The notions of tree, node in a tree, occurrence of a variable or a subtree in a tree, substitution, etc.... are supposed to be known (see [18]). We recall however those notions we shall use in the paper in order to fix the notations.

We use the Dewey notation for trees (see also [17]): nodes in a tree are denoted by finite words over the alphabet N, i.e. elements of the free monoid N^*.

Formally, a *tree* on F is a pair (D_t, t) consisting of:

a tree domain D_t which is a finite subset of $(N-\{0\})^*$ such that if $o = n_1 \ldots n_p$ is in D_t then (a) every left factor $o' = n_1 \ldots n_q$, $q \leqslant p$, also is in D_t, (b) for all $i \leqslant n_p$, $o'' = n_1 \ldots n_{p-1} i$ also is in D_t

a total mapping t from D_t into F such that, for any o in D_t, if $t(o) = f \in F$ and f is of rank p then o has exactly p "sons" (i.e. nodes o' of the form $o' = on_i$) in D_t.

In the sequel, D_t will be omitted and a tree will be denoted by t. A *node* in a tree is an element of D_t; $t(o)$ is called the *label* of node o; o is also called an *occurrence* of $t(o)$ in t; nodes or occurrences will be denoted by o. Note that D_t contains the empty word ε: the root of t.

Let t, t' be trees on F and o be a node in t; $t(t'/o)$ denotes the tree obtained by substituting t' for the node o in t and is defined by:

(a) $t(t'/o)(o\,o') = t'(o')$ for any o' in $D_{t'}$ and

(b) $t(t'/o)(o'') = t(o'')$ if o is not a left factor of o'' (recall that o is a left factor of o'' iff there exists an o' with $o'' = oo'$).

The subtree $t_{|o}$ of t (at occurrence o) is the tree t'' defined by: $t''(o') = t(oo')$ for any oo' in D_t.

The depth $d(t)$ of a tree t is defined by:

$d(t) = \sup\{|o| + 1 / o \in D_t\}$, where $|o|$ is the length of o considered as an element of N^*.

Let $A(F)$ denote the set of trees on F; a tree with variables in V is a tree on $F \cup V$; the set of trees with variables is denoted by $A(F \cup V)$ or, when we want to point out the variables, $A(F, V)$: intuitively, the variables are intended to range over a set of trees, e.g. $A(F)$ or $A(F, V)$. $A(F)$ (resp. $A(F, V)$), with the obvious algebraic structure, is called the *free F-magma*, or *F-algebra* (resp. the free F-magma over V), and is denoted, in the ADJ terminology T_F (resp. $T_F(V)$).

$A_i(F, V)$ denotes the trees having at most i variables, and $A^i(F)$ (resp. $A^i(F, V)$) denotes the trees of depth i; clearly, $\forall i$: $A^i(F) \subset A^i(F, V)$ and $A^i(F) \subset A(F) \subset A(F, V)$, and similar relations hold for the A_i's.

Throughout this paper we will use the word notation for trees. A tree is represented by a word over the alphabet $F \cup \{(,),c\}$, where c is the comma, as follows:

a stands for $a \in F_0$

$f(t_1,\ldots,t_r)$ stands for the tree having root $f \in F_r$ and direct subtrees $t_1,\ldots,t_r$, and would be pictured as:

A tree t in $A_p(F,V)$ will be denoted by $t(v_1,\ldots,v_p)$ in order to point out the variables; the shorthand vector notation $t(\vec{v})$ will be used; $t(t_1,\ldots,t_p)$ denotes the tree obtained by simultaneously substituting t_i for each occurrence of v_i in $t(v_1,\ldots,v_p)$, for $i=1,\ldots,p$; formally, $t(t_1,\ldots,t_p)$ is the image of $t(v_1,\ldots,v_p)$ by the $F \cup V$-algebra morphism h defined by: $h(v_i)=t_i$ for $i=1,\ldots,p$, and $h(f)=f$ for f in F. We will also use a vector shorthand notation $t(\vec{t})$ for $t(t_1,\ldots,t_p)$. Alternatively, when wanting to point out which trees are being substituted for which variables, we will note: $t(t_1/v_1,\ldots,t_p/v_p)$ (or $t(\vec{t}/\vec{v})$).

A *prefix* of $t \in A(F,V)$ is a tree $\underline{t} \in A(F,\{w_1,\ldots,w_s\})$, with $\{w_1,\ldots,w_s\}$ disjoint from V, $D_{\underline{t}} \subset D_t$, and such that there exist subtrees $t_1,\ldots,t_s$ of t with $t=\underline{t}(t_1/w_1,\ldots,t_s/w_s)$.

Recall finally that the yield of a tree is its image under the homomorphism h: $A(F) \to F_0^*$, where F_0^* is viewed as an F-algebra where all f in F_r, $r \geq 1$, are interpreted as concatenation. Formally, h is defined inductively by:

$h(a)=a$ for a in F_0

$h(f(t_1,\ldots,t_r))=h(t_1)\ldots h(t_r)$ for f in F_r, $r \geq 1$ and $t_1,\ldots,t_r$ in $A(F)$.

We will write yield(t) rather than $h(t)$.

These notations are illustrated in Figure 1 below. For more details on trees consult [18].

3. Pushdown Tree Automata and Context-Free Grammars

Definition 1. A *context-free tree grammar* $\mathcal{G}$ is a 4-tuple (F,Φ,P,G_0), where:

(a) F is a finite ranked alphabet (of terminal symbols)

(b) $\Phi=\{G_0,G_1,\ldots,G_n\}$ is a finite ranked alphabet of function variables (or nonterminal symbols)

(c) P is a finite set of pairs $\langle G_i(x_1,\ldots,x_{r(G_i)}),\ t_i^j\rangle$ where, for $i=0,\ldots,n$, $j=1,\ldots,n_i$, $t_i^j \in A(F \cup \Phi,\{x_1,\ldots,x_{r(G_i)}\})$

(d) G_0, the axiom or initial nonterminal, is a distinguished symbol of rank 0 in Φ.

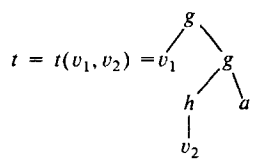
We will use a vector shorthand notation and abbreviate $G(x_1,\ldots,x_{r(G)})$ in $G(\vec{x})$. Each pair $\langle G_i(\vec{x}),t_i^j\rangle$ in P is called a *production* of $\mathcal{G}$, and is denoted by $G_i(\vec{x}) \to t_i^j$. Productions of $\mathcal{G}$ turn into rewriting rules: consider the x_k's as dummy variables (or parameters) standing for trees in $A(F)$. The triple (F,Φ,P) can thus be viewed as a nondeterministic recursive program scheme, or a tree rewriting system. Grouping together all right hand sides of productions having

the same nonterminal as left hand side, we get the following synthetic notation for a tree rewriting system:

$$G_i(\vec{x}) \to T_i = t_i^1 + \cdots + t_{t_i}^{n_i} \subset A\left(F \cup \Phi, \left\{x_1, \ldots, x_{r(G_i)}\right\}\right)$$

where + denotes the set theoretical union.

Immediate rewritings ($\underset{\mathcal{G}}{\Rightarrow}$) and derivations ($\overset{*}{\underset{\mathcal{G}}{\Rightarrow}}$) according to $\mathcal{G}$ are defined as usual (cf. [18]). Let us recall the definitions here for the reader's convenience. The immediate rewriting according to $\mathcal{G}$ is the relation $\underset{\mathcal{G}}{\Rightarrow}$ defined on $A(F \cup \Phi) \times A(F \cup \Phi)$ by: $t \underset{\mathcal{G}}{\Rightarrow} t'$ iff there exists some *prefix* $\underline{t}$ of t, an occurrence o in t labeled by the function variable G_i of rank s (i.e. $t(o) = G_i$), and subtrees $t_1, \ldots, t_s$ of t such that: $t = \underline{t}(G_i(t_1, \ldots, t_s)/o)$ and $t' = \underline{t}(t_i^j(t_1, \ldots, t_s)/o)$ for some t_i^j in T_i.

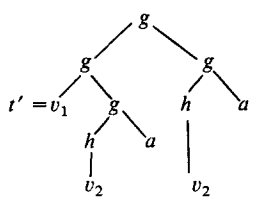


$t \in A(F, \{v_1, v_2\})$ with $F = \{a, h, g\}$
$r(a) = 0 \quad r(h) = 1 \quad r(g) = 2$

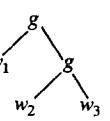
$D_t = \{\varepsilon, 1, 2, 21, 22, 211\}$

$d(t) = 4$ $\quad t(\varepsilon) = t(2) = g \quad t(1) = v_1 \quad t(21) = h$
$t(211) = v_2 \quad t(22) = a$

$t' = t(t/1) = t(t, v_2)$ can be drawn as:



yield $(t') = v_1 v_2 a v_2 a$.

Let: $\underline{t} =$ 

$\underline{t}$ is a prefix of t and t'.

Fig. 1

Intuitively, at occurrence o, G_i is macroexpanded, i.e. replaced by the right handside t_i^j of production $G_i(x_1,\ldots,x_s) \to t_i^j$; simultaneously, $t_1,\ldots,t_s$ are substituted for $x_1,\ldots,x_s$. Whenever no ambiguity can occur, the subscript $\mathcal{G}$ is omitted and $\underset{\mathcal{G}}{\Rightarrow}$ is denoted by $\Rightarrow$, as will be done in the sequel. The derivation relation $\overset{*}{\Rightarrow}$ according to $\mathcal{G}$ is the reflexive and transitive closure of $\Rightarrow$.

More operationally, we can view $\mathcal{G}$ as a semi-thue system with variables.

Recall from [11] that a semi-thue system with variables is a 4-tuple $\mathcal{S} = (A, X, D, R)$ with

A an alphabet of terminal symbols

X a set (disjoint from A) of variable symbols

D a mapping from X into the power set of A^*

$R \subset (A \cup X)^* \times (A \cup X)^*$ a finite set of rules denoted by $r: \varphi \to \psi$.

Each variable x in X is meant to range over $D(x)$ so, for each rule $r: \varphi \to \psi$ in R, $s(r)$ is defined to be the set of all rules $\alpha \to \beta$ in $A^* \times A^*$ such that there exists a homomorphism $h: (A \cup X)^* \to A^*$ with $h(x) \in D(x)$ for all x in X, $h(a) = a$ for all a in A, $h(\varphi) = \alpha$ and $h(\psi) = \beta$. Let $s(R) = \bigcup_{r \in R} s(r)$. $\underset{\mathcal{G}}{\Rightarrow}$ is then defined on $A^* \times A^*$ by, for any $\alpha \to \beta$ in $s(R)$, η and θ in A^*, $\eta\alpha\theta \underset{\mathcal{G}}{\Rightarrow} \eta\beta\theta$. $\overset{*}{\underset{\mathcal{G}}{\Rightarrow}}$ is the reflexive and transitive closure of $\underset{\mathcal{G}}{\Rightarrow}$.

Note now that $\mathcal{G}$ can be viewed as a semi-thue system with variables.

$$A = F \cup \Phi \cup \{(,),c\},$$

where $V = \{x_1,\ldots x_{\sup\{r(G_i)/i=1,\ldots,n\}}\}$ and c is the comma symbol.

$$X = V$$

$$D(x) = A(F \cup \Phi) \text{ for every } x \in X$$

$$R = P.$$

Then, $\overset{*}{\Rightarrow}$ is the derivation relation of the semi-thue system.

A derivation $t \Rightarrow t_1 \ldots \Rightarrow t_p$ consisting of p immediate rewritings will be denoted by $t \overset{p}{\Rightarrow} t_p$.

A derivation $t \overset{*}{\Rightarrow} t'$ is said to be OI, or outside-in [15, 13] iff for all immediate rewritings $t^k = \underline{t}(G_i(t_1,\ldots,t_s)/o) \Rightarrow t^{k+1} = \underline{t}(t_i^j(t_1,\ldots,t_s)/o)$ composing it, and for all left factors o' of o (i.e. $o = o'o''$), $t^k(o') \in F$; in other words, no ancestor of o is labeled by a function variable.

Recall [15, 13] that any t' in $A(F)$ such that $t \overset{*}{\Rightarrow} t'$ can also be obtained from t by an OI derivation.

The context-free tree language $L(\mathcal{G})$ generated by $\mathcal{G}$ is $L(\mathcal{G}) = \{t \in A(F)/G_0 \overset{*}{\Rightarrow} t\}$.

$\mathcal{G}$ also generates [12] a macro string language $L_s(\mathcal{G})$ obtained by concatenating the leaves of each tree in $L(\mathcal{G})$:

$L_s(\mathcal{G}) = \{\text{yield}(t)/t \in L(\mathcal{G})\} \subset F_0^*$. The $L_s(\mathcal{G})$'s coincide with the class of indexed languages [1, 12, 15].

Example 1. Let $\mathcal{G}$ be defined by: $F = \{a, g, h, f\}$, $r(a) = 0$, $r(g) = r(h) = 1$, $r(f) = 2$; $\Phi = \{G_0, K\}$, $r(K) = 1$ and $P = \{G_0 \to K(a), K(x) \to f(x, K(h(x)))+$

$g(x)\}$ then

$$L(\mathcal{G}) = \{g(a), f(a, g(h(a))), \ldots, f(a, f(h(a), \ldots, f(h^{n-1}(a), g(h^n(a)))\ldots)), \ldots\}$$

$$L_s(\mathcal{G}) = \{a^n / n \geq 1\}$$

$\mathcal{G}$ and $L(\mathcal{G})$ are displayed in a tree like form in Figure 2 below.

A pushdown tree automaton is a generalization of topdown tree automata [31, 32]: it is obtained by allowing an auxiliary storage consisting of a single tree. The root of this pushdown tree is always accessible, and according to the state and the input symbol scanned, this root can be popped or replaced by a tree. Like top-down tree recognizers, pushdown tree automata have the essential ability of duplicating themselves, and also their pushdown store, as they go down in the input tree. Bottom-up pushdown tree automata have also been investigated [28, 29]: they, of course, don't duplicate themselves, but their storage consists of finite sets of trees, whose roots must be simultaneously accessible; this is, in some sense, unavoidable, due to the inherent parallelism present in context-free tree languages.

Intuitively, PDTA's (pushdown tree automata) can be viewed as an extension of both:

- finite top down tree automata [31]: like usual (string) PDA's extend finite automata, PDTA's extend finite tree automata by allowing an auxiliary storage consisting of a tree and the possibility of ε-moves ignoring the input.
- usual PDA's by allowing trees instead of strings in both the input and the store: as in the PDA case, the finite state control processes both input and stack in a left-to-right (i.e. top down) manner; namely the machine can read only the root of each input and stack tree, and processes the stack by substituting a tree for the root (pushing) or deleting the root and selecting one of its sons (popping) (see Figure 3).

Definition 2. A *pushdown tree automaton* (PDTA) M is a six-tuple (Q, F, Π, q_0, Z_0, R), where:

Q is a finite set of states

F is a finite ranked alphabet, called the input alphabet

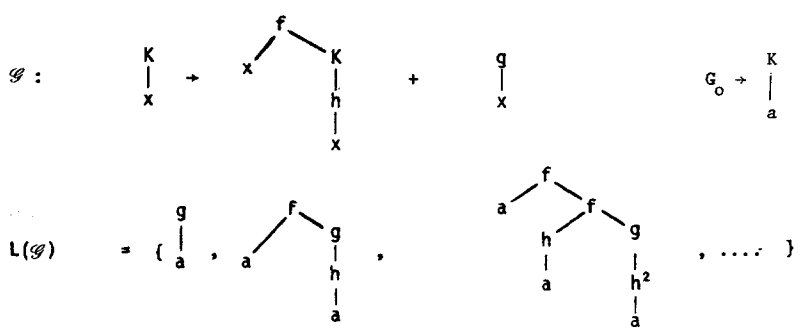


Fig. 2

Π is a finite ranked alphabet, called the pushdown alphabet. We may suppose, w.l.o.g. that Π and F are disjoint alphabets.
q_0 is the initial state, $q_0 \in Q$
Z_0 is the start symbol, appearing initially on the pushdown store, $Z_0 \in \Pi_0$.
R is a finite set of rules (or moves) of the form:
(i) read rule

$$q(f(v_1,\ldots,v_r), E(x_1,\ldots,x_s)) \to f(q_1(v_1,\pi_1),\ldots,q_r(v_r,\pi_r))$$

where

$$f \in F_r, E \in \Pi_s, \pi_i \in A(\Pi, \{x_1,\ldots,x_s\}) \quad \text{for } i = 1,\ldots,r, \quad q, q_i \in Q.$$

(ii) ε-rule

$$q(v, E(x_1,\ldots,x_s)) \to q'(v,\pi')$$

where $E \in \Pi_s, \pi' \in A(\Pi, \{x_1,\ldots,x_s\}), q, q' \in Q, v$ ranges over $A(F)$.

A PDTA M is said to be *deterministic* (DPDTA) iff for any pair (q, E), either there exists at most one ε-rule in state q with pushdown root E and no read move, or there exists no ε-rule and at most one read rule for each f in F (in state q with pushdown root E).

An *instantaneous description* (ID) of M is a triple $q(t,\pi) \in Q \times A(F) \times A(\Pi)$. If $t = f(t_1,\ldots,t_n)$ and $\pi = E(\pi_1,\ldots,\pi_s)$, then M is currently in state q, reading input symbol f and scanning root (top) of stack E. An initial ID is of the form

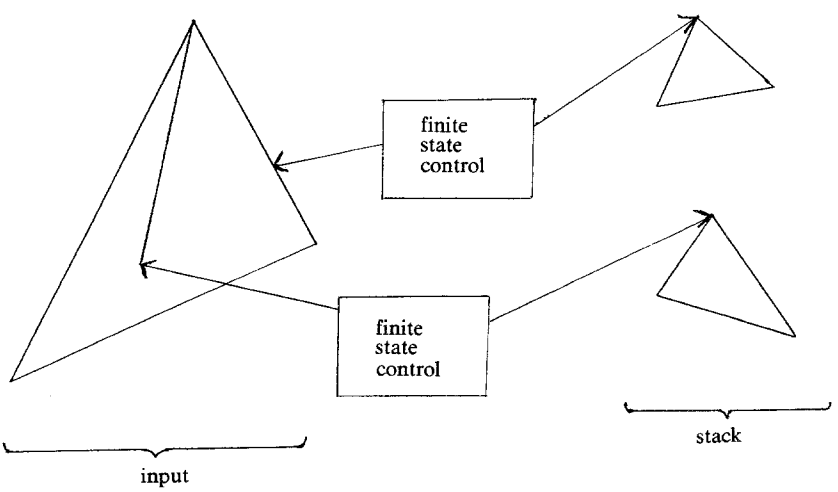


Fig. 3

$q_0(t, Z_0)$. ID's might also be called configurations [6]. We here follow the terminology of [19].

Notation. Elements of Π are denoted by $B, C, D, \ldots$ (symbols of rank 0), $E, G, H, K, \ldots$ (symbols of rank ≥ 1); $x_1, x_2, \ldots$ are variables representing trees of $A(\Pi)$. Elements of F are denoted by the corresponding lower-case letters ($b, c, d, \ldots e, f, g, h, k, \ldots$ and $u, v, v_1, v_2, \ldots$ for variables). Trees (or terms) in $A(F)$ (resp. $A(\Pi)$) are denoted by t, t', t_i (resp. π, π', π_i). Variables v_i's (resp. x_i's) are dummy variable symbols intended to represent positions (or parameters) to be replaced by elements ranging over $A(F)$ (resp. $A(\Pi)$) in the set of instantaneous descriptions of M.

A *configuration* c of M is an element of the form $t(id_1, \ldots, id_n)$, where $t \in A(F, \{v_1, \ldots, v_n\})$ and $id_1, \ldots, id_n$ are ID's of M, i.e. $c \in A(F, Q \times A(F) \times A(\Pi))$.

The *move relation* $\vdash$ of M is the relation defined on configurations by:

$$c = t(id_1, \ldots, id_{i-1}, id_i, id_{i+1}, \ldots, id_n) \vdash c'$$
$$= t(id_1, \ldots, id_{i-1}, c_i, id_{i+1}, \ldots, id_n)$$

iff either $id_i = q_i(f(t_1, \ldots, t_r), E(\pi_1', \ldots, \pi_s'))$
and $c_i = f(q_1(t_1, \pi_1(\vec{\pi}'/\vec{x})), \ldots, q_r(t_r, \pi_r(\vec{\pi}'/\vec{x})))$
where $q_i(f(v_1, \ldots, v_r), E(x_1, \ldots, x_s)) \to f(q_1(v_1, \pi_1), \ldots, q_r(v_r, \pi_r))$ is a read rule of M, $\vec{\pi}' = (\pi_1', \ldots, \pi_s')$ and $\vec{x} = (x_1, \ldots, x_s)$

or $id_i = q_i\big(t, E\big(\pi_1', \ldots, \pi_s'\big)\big)$ and $c_i = q'\big(t, \pi\big(\vec{\pi}'/\vec{x}\big)\big)$

where $q_i(v, E(x_1, \ldots, x_s)) \to q'(v, \pi)$ is an ε-rule of M, $\vec{\pi}' = (\pi_1', \ldots, \pi_s')$ and $\vec{x} = (x_1, \ldots, x_s)$.

Let the *computation relation* $\overset{*}{\vdash}$ of M be the reflexive and transitive closure of $\vdash$.

A *sentential form* of M is a configuration $c = t'(id_1, \ldots, id_n)$ such that, for some t in $A(F)$, $q_0(t, Z_0) \overset{*}{\vdash} c$. Notice that, if $id_i = q_i(t_i, \pi_i)$ for $i = 1, \ldots, n$, then $t = t'(t_1, \ldots, t_n)$.

A PDTA M is said to be *real-time* (resp. *quasireal-time*) iff it has no ε-rule (resp. a bounded number of consecutive ε-moves).

Remark 1. Note that, if we consider all states to have rank 2, left and right-hand sides of the rules (i) and (ii) in definition 2 are trees; hence these rules can be viewed as ordinary tree rewriting rules. Namely, we can view M as a semi-thue system with variables

$M = (\mathscr{A}, \mathscr{X}, \mathscr{D}, \mathscr{R})$, where:
$\mathscr{A} = F \cup \Pi \cup Q \cup \{(,), c\}$ where c is the comma.
$\mathscr{X} = X \cup V$ where $X = \{x_1, x_2, \ldots\}$ and $V = \{v, v_1, v_2, \ldots\}$
$\mathscr{D}(v) = A(F)$, $\mathscr{D}(x) = A(\Pi)$, for all v in V, x in X
$\mathscr{R} = R$

Then, $\vdash$ coincides with $\Rightarrow$. The sentential forms of M thus coincide with the sentential forms of the rewriting system.

Remark 2. Note that, in read rules, in the special case that $r = 0$, the rule is $q(a, E(x_1, \ldots, x_s)) \to a$. The current stack is thus deleted; this means intuitively that "q is a final state with respect to a and E"; namely final states are implicitly present: the automaton knows it has successfully completed a computation when it reads a symbol of rank 0. This naturally leads to acceptance by "final states" rather than by "empty store".

Definition 3. The tree language accepted (by final state) by M is defined by:

$$T(M) = \{ t \in A(F)/q_0(t, Z_0) \vdash^{*} t \}.$$

M is said to accept by empty store (and final state) iff all read rules (i) which read an input symbol of rank 0 are of the form: $q(a, B) \to a$, i.e. both $f = a$ and $E = B$ are of rank 0.

Thus, intuitively acceptance occurs whenever M processes the whole of t; if moreover read moves on leaves always result in popping a leaf of the push down M accepts by empty store.

Example 2. Let M be defined by
$Q = \{q\}$, $F = \{a, g, h, f\}$ as in example 1,
$\Pi = \{Z_0, B, H, K\}$, $r(Z_0) = r(B) = 0$ and $r(H) = r(K) = 1$, $q_0 = q$,
$Z_0 = Z_0$, and R is the following set of rules (all of type (i)):
$q(g(u), Z_0) \to g(q(u, B))$
$q(f(u, v), Z_0) \to f(q(u, B), q(v, K(H(B))))$
$q(f(u, v), K(x)) \to f(q(u, x), q(v, K(H(x))))$
$q(a, B) \to a$
$q(h(u), H(x)) \to h(q(u, x))$
$q(g(u), K(x)) \to g(q(u, x))$.
This automaton recognizes the language $L(\mathcal{G})$ of example 1. Moreover, this is an example of a real-time (without ε-rules), deterministic, and restricted automaton (i.e. its pushdown alphabet consists only of words, see definition 4 of section 5) which accepts by empty store.

Proposition 1. *If a tree language is accepted by some* PDTA M, *it can be accepted by empty store by some* PDTA M'.

Proof. M' will simulate M, but will delay reading the leaves by making a "guess" stored in the state and checking the guess while emptying the stack. Formally, M' has the same alphabets as M; its set of states Q' contains Q together with new states q^a for each q in Q such that M has a "leaf-reading" rule $q(a, E(x_1, \ldots, x_s)) \to a$. The rules of M' are then obtained by adding to the rules of M the following set of rules:

for each rule $q(a, E(x_1, \ldots, x_s)) \to a$ of M, add

the set of ε-rules (popping the stack)

$$q(v, E(x_1,\ldots,x_s)) \to q^a(v, x_1)$$
$$q^a(v, G(x_1,\ldots,x_s)) \to q^a(v, x_1) \quad \forall G \in \Pi_s, s \geq 1$$

the set of read-rules

$$q^a(a, B) \to a \quad \forall B \in \Pi_0$$ □

Several variants of PDTA's which do not extend the class of defined languages may be introduced: e.g. several initial states, a starting tree or a set of starting trees (of depth possibly greater than two) instead of a start symbol of rank 0, the ability to read in a single move input trees of depth greater than two, etc.... These variants do not give additional power to the corresponding automata. Let us check, for instance, the following two propositions, which will be useful in proving the equivalence of PDTA's and indexed grammars (section 5).

Proposition 2. *Any language accepted by a* PDTA *with additional generalized rules of the form*:

(iii) $q(f(v_1,\ldots,v_r), x) \to f(q_1(v_1, \pi_1),\ldots,q_r(v_r, \pi_r))$

(iv) $q(v, x) \to q'(v, \pi')$

with $\pi', \pi_1,\ldots,\pi_r$ *in* $A(\Pi, \{x\})$, *and* x *ranging over* $A(\Pi)$, *can be accepted by a normal* PDTA.

Proof. Note first that the move relation corresponding to, e.g. a type (iv) ε-rule, is the following: $id = q(t, \pi) \vdash q'(t, \pi'(\pi))$, with $t \in A(F)$, $\pi \in A(\Pi)$. It is then clear that any type (iv) ε-rule can be simulated by the following set of type (ii) ε-rules:

$$q(v, E(x_1,\ldots,x_s)) \to q'(v, \pi'(E(x_1,\ldots,x_s))), \quad \text{for any} \quad E \text{ in } \Pi.$$

Similarly, any type (iii) rule can be simulated by the set of type (i) read rules:

$$q(f(v_1,\ldots,v_r), E(\vec{x})) \to f(q_1(v_1, \pi_1(E(\vec{x}))),\ldots,q_r(v_r, \pi_r(E(\vec{x}))))$$

for any E in Π. □

Proposition 3. *Let language* L *be accepted by a* PDTA M *with extended read rules of the form*:

(v) $q(t(v_1,\ldots,v_r), E(x_1,\ldots,x_s)) \to t(q_1(v_1, \pi_1),\ldots,q_r(v_r, \pi_r))$

with E, π_i *as in definition* 2, *and* t *is in* $A(F, \{v_1,\ldots,v_r\})$, *each variable* v_i *occurs at most once in* t, t *has depth* ≥ 2.

Then L *can be accepted by a normal* PDTA M'.

Proof. Let $m: q(t(v_1,\ldots,v_r), E(x_1,\ldots,x_s)) \to t(q_1(v_1,\pi_1),\ldots,q_r(v_r,\pi_r))$ be a type (v) rule of M. We will simulate m by a sequence of read rules of M'. For each such m let $O_m = \{o/o$ is an occurrence in $t(v_1,\ldots,v_r)$ such that $t(v_1,\ldots,v_r)(o) \in F\}$; O_m is thus the set of occurrences not labeled by variables in t. Then, if $M = (Q, F, \Pi, q_0, Z_0, R)$, $M' = (Q', F, \Pi, q_0, Z_0, R')$ where:

$Q' = Q \cup \{(m,o)/m$ is an extended read rule of M and o is in $O_m\}$

R' is defined as follows:

(1) type (i) and (ii) rules of R are in R'

(2) each type (v) rule m of R is replaced by the following set of type (i) rules:

for each o in O_m, let $f_0 = t(v_1,\ldots,v_r)(o) \in F_n$, $n \geq 0$, be the label of occurrence o in $t(v_1,\ldots,v_r)$; then R' contains the rule:

$$p(f_0(v_1,\ldots,v_n), E(x_1,\ldots,x_s)) \to f_0(id_1,\ldots,id_n)$$

with

$$p = \begin{cases} q & \text{if} \quad o = \varepsilon \\ (m,o) & \text{if} \quad o \neq \varepsilon \end{cases}$$

and for each j:

$$id_j = \begin{cases} (m,oj)(v_j, E(x_1,\ldots,x_s)) & \text{if } oj \in O_m \\ q_k(v_j,\pi_k) & \text{if} \quad oj \notin O_m \quad \text{and} \quad t(v_1,\ldots,v_r)(oj) = v_k \end{cases}$$

It should be clear that $T(M) = T(M')$. □

Remark 3 ([9]). On the other hand, allowing PDTA's to read in a single move pushdown trees of depth > 2 (i.e. a "look-ahead" on the store which is operated as a stack rather than a pushdown) indeed results in a much more powerful device as soon as the pushdown alphabet X contains a symbol of rank > 1. Let e.g. G be a binary symbol in X, then one can simulate a Turing machine with a look-ahead of depth 1 on the store: the pushdown $G(\pi_1, \pi_2)$ can be viewed as two pushdowns, connected by G representing the Turing machine head; π_1 (resp. π_2) stands for the part of the Turing machine tape lying to the left (resp. right) of the head. Since, as we shall see in theorem 1 below, PDTA's accept only context-free tree languages, such a look-ahead on the stack would vastly increase the class of accepted languages.

Proposition 4. PDTA's *accept the class of creative dendrolanguages* [26].

Sketch of Proof. When dropping all first arguments of states in rules (ii), (v) one obtains rewriting rules exactly similar to those of the creative dendrogrammars. □

Note that it is proved in Rounds' paper (theorem 7 of [25], see also [1]) that the number of states of a creative dendrogrammar can always be reduced to 1,

thereby proving that creative and context-free languages coincide. However, both Rounds and Boudol's papers [25, 7] stress the grammatical aspects of the problem, whereas we want to consider its operational and automaton theoretical aspects. This is the reason why:

1. We will give in Theorem 1 below a complete proof that PDTA's recognize context-free tree languages. Proposition 4 will then follow from Theorem 1. The spirit of our proof will be quite similar to the one in Rounds paper (Theorem 7).

2. We will later prove a more operational simplification theorem (Theorem 3 below), more remote from the considerations in [7, 25], by reducing the pushdown to a linear one, i.e. simplifying the store instead of reducing the number of states. This leads to a simple proof of the equivalence of yields of context-free tree languages and indexed string languages.

Theorem 1. *A tree language is context-free if and only if it can be accepted by a* PDTA.

Proof. "Only if" part: Let $\mathcal{G} = (F, \Phi, P, G_0)$ be a context-free tree grammar. For each production $G_i(x_1,\ldots,x_s) \to t_i^j$, decompose t_i^j in the form: $t_i^j = t_{ij}(\theta_1,\ldots,\theta_n)$ with $t_{ij} \in A(F, \{v_1,\ldots,v_n\})$, $n \geq 0$, $\forall i = 1,\ldots,n$, $\theta_i \in A(F \cup \Phi, X_s)$ and $\theta_i(\varepsilon) \in \Phi \cup X_s$ where $X_s = \{x_1,\ldots,x_s\}$ (i.e. t_{ij} is a prefix of t_i^j and the root of each θ_i is a function variable or an x_i, namely we mark outermost occurrences of function variables in t_i^j).

Notice that, in case $t_i^j(\varepsilon) \in \Phi \cup X_s$ the above decomposition reduces to $t_i^j = v(t_i^j)$.

Now let M be the automaton having the single state q, input alphabet F, pushdown alphabet $\Pi = F \cup \Phi$, start symbol $Z_0 = G_0$ and rules R defined by:

for each production $G_i(x_1,\ldots,x_s) \to t_i^j$ in P, R contains the rule:

$$\text{(vi)}\ q\big(t_{ij}(v_1,\ldots,v_n), G_i(x_1,\ldots,x_s)\big) \to t_{ij}\big(q(v_1,\theta_1),\ldots,q(v_n,\theta_n)\big)$$

for each terminal f in F_n, R contains the rule:

$$\text{(vii)}\ q\big(f(v_1,\ldots,v_n), f(x_1,\ldots,x_n)\big) \to f\big(q(v_1,x_1),\ldots,q(v_n,x_n)\big)$$

Note that:

1. R possibly contains extended type (v) read rules

2. in the special case when the root of t_i^j is labeled by a (function) variable, taking $t_{ij} = v$ in the above rule (vi) leads to a type (ii) ε-rule.

Now the following fact is clear and gives the desired result:

Fact 1. There exists an OI derivation sequence of $\mathcal{G}$: $G_i(t_1,\ldots,t_s) \overset{*}{\Rightarrow} t' \in A(F)$ iff there exists a successful computation sequence of M: $q(t', G_i(t_1,\ldots,t_s)) \overset{*}{\vdash} t'$ (with t_i in $A(F \cup \Phi)$ for $i = 1,\ldots,s$).

This fact shows that $T(M) = L(\mathcal{G})$ since it is well known that any tree in $L(\mathcal{G})$ can be obtained by an OI derivation [12, 15]; this is also one of the reasons

in choosing a top-down behavior for our PDTA's in contrast to the TPDA's of [29]: by sticking more closely to the grammatical standpoint we obtain simpler automata.

"if" part: Let $M = (Q, F, \Pi, q_0, Z_0, R)$

The construction of a grammar G generating $T(M)$ is somewhat more complicated. We first give the idea of it: essentially, the changes of the pushdown store will be modeled by the productions of the grammar, but we will have also to model the additional ability of changing the state; hence, the nonterminals of $\mathcal{G}$ will encode pairs (state, top pushdown symbol). Moreover, the arguments of each nonterminal (q, K) should render all possible "next states" after K has been popped. Formally, let $k = \text{card}(Q)$, and $\Phi = \{G^q / q \in Q, G \in \Pi\}$ where each G^q in Φ has rank $k \times r(G)$; similarly, if X is the set of pushdown variable symbols of M, let $X^Q = \{x^q / x \in X, q \in Q\}$ be a set of variables (x^q is intuitively intended to correspond to the selection of variable x and next state q). Define σ by: σ: $Q \times A(\Pi, X) \to A(\Phi, X^Q)$ satisfies $\sigma(q, s) = s^q$ if s has rank 0 or is a variable, and, using a vector notation $\vec{\sigma}(t)$ instead of $\sigma(q_1, t), \sigma(q_2, t), \ldots, \sigma(q_k, t)$, define by induction: $\sigma(q, G(t_1, \ldots, t_{r(G)})) = G^q(\vec{\sigma}(t_1), \ldots, \vec{\sigma}(t_{r(G)}))$ (in short $\sigma(q, G(\vec{t})) = G^q(\vec{\sigma}(\vec{t}))$).

The grammar $\mathcal{G}$ generating $T(M)$ is now defined as follows: $\mathcal{G}$ has terminal alphabet F and nonterminal alphabet Φ, axiom $G_0 = \sigma(q_0, Z_0)$ and productions:

$\sigma(q, G(x_1, \ldots, x_s)) \to \sigma(q', \pi')$ for each ε-rule $q(v, G(x_1, \ldots, x_s)) \to q'(v, \pi')$ of R

$\sigma(q, G(x_1, \ldots, x_s)) \to f(\sigma(q_1, \pi_1), \ldots, \sigma(q_r, \pi_r))$ for each read rule $q(f(v_1, \ldots, v_r), G(x_1, \ldots, x_s)) \to f(q_1(v_1, \pi_1), \ldots, q_r(v_r, \pi_r))$ of R.

Notice that, if M is real-time, it has no ε-rule, hence $\mathcal{G}$ is Greibach. (See section 4 for the definition of Greibach grammar).

We now can state:

Fact 2. For every π in $A(\Pi)$, t in $A(F)$ and q in Q: $\sigma(q, \pi) \overset{n}{\Rightarrow} t$ by an OI derivation sequence of $\mathcal{G}$ of length n iff there exists a successful computation sequence of length n $q(t, \pi) \overset{n}{\vdash} t$ of M.

Proof. By induction on n. We first prove the "only if" part.

$n = 1$ then, forcibly, $\pi = G(\psi_1, \ldots, \psi_s)$ and $\sigma(q, \pi) \Rightarrow a \in F_0$ which can occur iff $q(a, \pi) \to a$ is a rule of M, hence gives a successful computation sequence of length 1

Inductive step: suppose $\sigma(q, \pi) \overset{n+1}{\Rightarrow} t$, then $\pi = G(\psi_1, \ldots, \psi_s)$ and:

either $\sigma(q, \pi) \Rightarrow \sigma(q', \pi'(\psi_1, \ldots, \psi_s)) \overset{n}{\Rightarrow} t$ and hence, by the construction of $\mathcal{G}$, there exists an ε-rule $q(v, G(x_1, \ldots, x_s)) \to q'(v, \pi')$ of M, and by the induction hypothesis, there exists a length n computation sequence $q'(t, \pi'(\psi_1, \ldots, \psi_s)) \overset{n}{\vdash} t$ of M; whence the successful computation sequence of length $n+1$: $q(t, \pi) \vdash q'(t, \pi'(\psi_1, \ldots, \psi_s)) \overset{n}{\vdash} t$.

or $\sigma(q,\pi) \Rightarrow f(\sigma(q_1,\psi_1'),\ldots,\sigma(q_r,\psi_r'))$ and

a) $t = f(t_1,\ldots,t_r)$

b) for each $i=1,\ldots,r$, $\sigma(q_i,\psi_i') \overset{n_i}{\Rightarrow} t_i$ by an OI derivation sequence and

$\sum_{i=1}^{r} n_i = n$

then, there exists a read rule

$$q(f(v_1,\ldots,v_r),G(x_1,\ldots,x_s)) \to f(q_1(v_1,\pi_1),\ldots,q_r(v_r,\pi_r)) \quad \text{of } M,$$

and, by the inductive hypothesis, there exist r successful computation sequences: $q_i(t_i,\psi_i') \overset{n_i}{\vdash} t_i$, for $i=1,\ldots,r$. Whence the successful computation sequence: $q(f(t_1,\ldots,t_r), G(\psi_1,\ldots,\psi_s)) \vdash f(q_1(t_1,\psi_1'),\ldots,q_r(t_r,\psi_r')) \overset{n_1}{\vdash} f(t_1,q_2(t_2,\psi_2'),\ldots,q_r(t_r,\psi_r')) \overset{n_2}{\vdash} \cdots \overset{n_r}{\vdash} t = f(t_1,\ldots,t_r)$ of length $n+1$.

A very similar proof shows that, if there exists a successful computation sequence $q(t,\pi) \overset{n+1}{\vdash} t$, we can translate it into an OI derivation sequence $\sigma(q,\pi) \overset{n+1}{\Rightarrow} t$. □

Fact 2 shows that $\mathcal{G}$, with axiom $\sigma(q_0, Z_0)$, generates $T(M)$. □

Example 3 (of the previous construction). Let M be defined by: $Q = \{q_0, q_1, q_2\}$, $F = \{b, c_1, c_2\}$, $\Pi = \{G, C, Z_0\}$ with $r(b)=2$, $r(G)=1$, $r(C)=r(c_1)=r(c_2)=0$. The rules are the following:

(0) $q_0(v, Z_0) \to q_0(v, G(C))$

(1) $q_0(v, G(x)) \to q_0(v, G^2(x))$

(2) $q_0(b(u,v), G(x)) \to b(q_1(u,x), q_2(v,x))$

(3) for $i=1,2$, $q_i(b(u,v), G(x)) \to b(q_i(u,x), q_i(v,x))$

(4) $q_i(c_i, C) \to c_i$ for $i=1,2$

It is associated with $\mathcal{G}$ having nonterminals G^0, G^1, G^2, of rank 3, Z_0^0, C^0, C^1, C^2, of rank 0, axiom Z_0^0, and productions:

(0) $Z_0^0 \to G^0(C^0, C^1, C^2)$

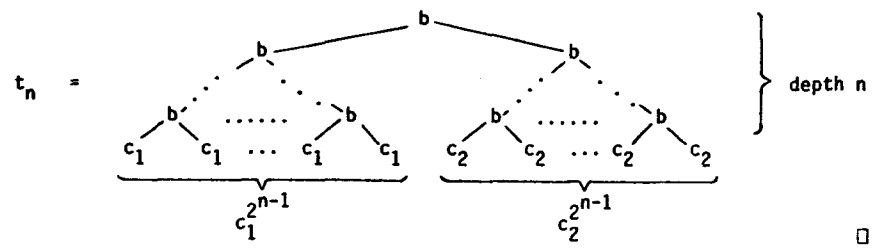
(1) $G^0(x^0,x^1,x^2) \to G^0(G^0(x^0,x^1,x^2), G^1(x^0,x^1,x^2), G^2(x^0,x^1,x^2))$

(2) $G^0(x^0,x^1,x^2) \to b(x^1,x^2)$

(3) for $i=1,2$, $G^i(x^0,x^1,x^2) \to b(x^i,x^i)$

(4) for $i=1,2$, $C^i \to c_i$

and $L(\mathcal{G}) = T(M)$ is the set of binary trees of the form:



□

Let us give as an example of application of this automaton theoretic characterization of context-free tree languages a very simple proof of the following:

Proposition. *The intersection of a context-free and a regular tree language is context-free.*

Proof. By a product construction used in many other proofs. □

On the other hand, if we already presume the above closure result and closure under linear (or alphabetic) erasing homomorphism, which are both special cases of closure under linear top-down transductions shown in [25, 26], a very elegant proof (due to J. Engelfriet) can be given to show that every $T(M)$ can be generated by a context-free tree grammar; let $\mathcal{G}' = (F', \Pi, R', Z_0)$ be the context-free grammar defined by:

each type (i) rule of M is translated into the production $E(x_1, \ldots, x_s) \to f^q_{(q_1, \ldots, q_r)}(\pi_1, \ldots, \pi_r)$ of R'

each type (ii) rule of M is translated into the production $E(x_1, \ldots, x_s) \to d^q_{q'}(\pi')$

where $f^q_{(q_1, \ldots, q_r)}$ is a new rank r symbol in F', for every type (i) rule, and $d^q_{q'}$ is a new rank 1 symbol in F', for every type (ii) rule (d stands for "dummy"). Let L be the context-free language generated by $\mathcal{G}'$. To obtain $T(M)$ from L, it suffices:

a) to intersect L with a recognizable tree language L' (checking that everywhere the subscript of the father is the sequence of superscripts of its sons)

b) to then erase all symbols $d^q_{q'}$ and rename each $f^q_{(q_1, \ldots, q_r)}$ to f by a linear homomorphism h_e.

This can be formalized in the:

Lemma. *For any t in $A(F)$ and π in $A(\Pi)$, there exists a computation sequence $q(t, \pi) \overset{n}{\vdash} t$ iff there exists an* OI *derivation $\pi \overset{n}{\Rightarrow} t' \in A(F')$ of $\mathcal{G}'$ such that $t' \in L'$ and $h_e(t') = t$.*

4. Classes of Languages Recognized by PDTA's.

We shall now characterize several known classes of tree languages by the automata which accept them and see that these results nicely extend string PDA theory in most cases and are more complicated in some cases.

Let us say that a tree language L is a *real-time* (resp. *quasireal-time, deterministic*, etc...) iff it can be accepted by a real-time (resp. quasireal-time, deterministic, etc...) PDTA M.

Note that it would be seemingly more restrictive to demand that L be accepted by empty store by some (quasi)real-time M: this stems from the fact that in proposition 1, we need extra ε-moves in going from M to M' and that it might possibly take an unbounded number of ε-moves to pop a leftover stack.

Let us recall first some terminology. A context-free tree grammar $\mathcal{G} = (F, \Phi, P, G_0)$ is said to be:

a) in *Greibach normal form* (in short *Greibach*) iff the root of the right hand side of each production is a terminal symbol in F [4]. Greibach grammars are called strict Greibach in [5] who allow for rules $G(\vec{x}) \to x_i$ in Greibach grammars.

b) *Deterministic* iff it is Greibach, and moreover, for each left-hand side there exists at most one production whose right-hand side has a given root f in F [8]. Such grammars are the natural extension to trees of simple deterministic grammars [20]. We call them deterministic because the languages they generate are accepted by deterministic PDTA's (Proposition 6 and Theorem 2 below).

Proposition 5. *L is realtime (resp. realtime deterministic) iff it can be generated by a Greibach (resp. deterministic) grammar.*

Proof. "only if" part: the construction of the grammar $\mathcal{G}$ generating $T(M)$ given in theorem 1 clearly shows that if M is real-time (deterministic) then $\mathcal{G}$ is Greibach (deterministic).

"if" part: to any Greibach (deterministic) grammar $\mathcal{G}$ we can easily associate a "normal" Greibach (deterministic) grammar $\mathcal{G}'$ which

generates the same language as $\mathcal{G}$

is Greibach (deterministic)

has all its productions in the form:

$G(x_1,\ldots,x_s) \to f(\theta_1,\ldots,\theta_r)$ with $f \in F_r$ and, for $i = 1,\ldots,r$, $\theta_i \in A(F \cup \Phi, \{x_1,\ldots,x_s\})$ and $\theta_i(\varepsilon) \in \Phi \cup \{x_1,\ldots,x_s\}$ (the root of θ_i is labeled by a (function) variable).

For such a $\mathcal{G}'$ the construction given in the first part of Theorem 1 clearly defines a realtime (deterministic) automaton M. □

Proposition 6. *Any deterministic language is real-time.*

Proof. Let L be accepted by a deterministic M and $\mathcal{G}$ be the grammar associated with M in the construction of Theorem 1; the set Φ of function variables (or nonterminals) of $\mathcal{G}$ can be partitioned in two disjoint sets:

(i) nonterminals coming from read rules of M which lead to Greibach productions

(a) $G_i(x_1,\ldots,x_s) \to t_i^1 + \cdots + t_i^{n_i}$

with, for $j = 1,\ldots,n_i$, $t_i^j(\varepsilon) = f_j \in F$ and $j \neq j' \Rightarrow f_j \neq f_{j'}$.

(i.e. the type (a) nonterminals and associated productions define a deterministic grammar)

(ii) nonterminals coming from ε-rules which lead to productions:

(b) $G_i(x_1,\ldots,x_s) \to G_i'(\theta_1,\ldots,\theta_{s'})$

with, for $j = 1,\ldots,s', \theta_j \in A(F \cup \Phi, \{x_1,\ldots,x_s\})$.

Moreover, M being deterministic, if G_i is the left-hand side of a type (b) production, this will be the unique production in $\mathcal{G}$ having G_i as left-hand side; this will help in getting rid of all type (b) productions, because in such a grammar, we can easily:

1. Delete all circular nonterminals, namely nonterminals such that there exists a derivation $G(x_1,\ldots,x_s) \overset{*}{\Rightarrow} G(\theta_1,\ldots,\theta_s)$, while preserving the type (a) and (b) partitioning of productions, see also [4, 21] where such nonterminals are called left-recursive): any circular G is, in this special case, useless, i.e. cannot generate any tree in $A(F)$.

2. then, because of the "deterministic" property of $\mathcal{G}$, for any type (b) nonterminal G_i, there exists exactly one finite derivation sequence:

$$G_i(x_1,\ldots,x_s) \overset{p_1}{\Rightarrow} G_{i_1}(\theta_1^1,\ldots,\theta_{s_1}^1) \overset{p_2}{\Rightarrow} \cdots \overset{p_k}{\Rightarrow} G_{i_k}(\theta_1,\ldots,\theta_{s_k})$$

such that: for $j=1,\ldots k$, p_j is a type (b) production and G_{i_k} is a type (a) nonterminal. Then, the type (b) production associated with G_i can be replaced by the type (a) productions

$$G_i(x_1,\ldots,x_s) \to t_{i_k}^1(\theta_1,\ldots,\theta_{s_k}) + \cdots + t_{i_k}^{n_{i_k}}(\theta_1,\ldots,\theta_{s_k})$$

All type (b) productions can thus be deleted and we hereby obtain a deterministic Greibach grammar generating L. □

Remark 4. Obviously, not every real-time language is deterministic, e.g. $L = \{f(a,a), f(b,b)\}$ is realtime and not deterministic.

Corollary 1.

(i) *Every* DPDTA *is equivalent to a realtime* DPDTA.

(ii) *Not every* PDTA *is equivalent to a real-time* PDTA.

Proof.

(i) is a restatement of proposition 6.

(ii) results from proposition 5 and the fact that not every context-free tree language admits a grammar in Greibach normal form [21, 23]. □

This shows that the situation is somewhat more complex than in the case of context-free string languages and is further illustrated by the next proposition.

Proposition 7. *Any language accepted by a quasi-real-time* PDTA *can be generated by a Greibach grammar.*

Proof. Let $M=(Q,F,\Pi,q_0,Z_0,R)$ be a PDTA accepting L such that any computation sequence of M has at most k consecutive ε-moves.

Lemma 1. *L can be accepted by a* PDTA *M' such that:*

(i) *each sequence of consecutive ε-moves in any computation sequence of M' has length exactly k*

(ii) *in any computation sequence of M', between any two read rules there is a sequence of k ε-moves.*

Proof. Intuitively, the idea is to count the number of ε-moves in the state and force it to be exactly k. Formally, $M'=(Q',F,\Pi,q_0,Z_0,R')$, where $Q'=\{(q,i)/q\in Q, i\in\{0,\ldots,k\}\}$

R' is defined by:
for each type (ii) ε-rule of M, $q(v, E(x_1,\ldots,x_s)) \to q'(v, \pi')$, R' contains the ε-rules:

$$(q,i)(v, E(x_1,\ldots,x_s)) \to (q', i+1)(v, \pi') \qquad 0 \le i < k$$
$$(q,i)(v, E(x_1,\ldots,x_s)) \to (q, i+1)(v, E(x_1,\ldots,x_s)) \qquad 0 \le i < k-1$$

for each type (i) read rule of M

$$q(f(v_1,\ldots,v_r), E(x_1,\ldots,x_s)) \to f(q_1(v_1,\pi_1),\ldots,q_r(v_r,\pi_r))$$

R' contains the read rule:

$$(q,k)(f(v_1,\ldots,v_r), E(x_1,\ldots,x_s)) \to f((q_1,0)(v_1,\pi_1),\ldots,(q_r,0)(v_r,\pi_r))$$

and the set of ε-rules:

$$(q,i)(v, E(x_1,\ldots,x_s)) \to (q, i+1)(v, E(x_1,\ldots,x_s)) \qquad 0 \le i < k. \square$$

Lemma 2. *Let $M = (Q, F, \Pi, q_0, Z_0, R)$ be a* PDTA; *then there exists a real time* PDTA $M'' = (Q, F \cup \{d\}, \Pi, q_0, Z_0, R'')$ *such that $T(M) = \varphi(T(M''))$ where φ is the linear homomorphism $A(F \cup \{d\}) \to A(F)$ defined by*:

$$\varphi(f(t_1,\ldots,t_r)) = f(\varphi(t_1),\ldots,\varphi(t_r)) \quad \text{for } f \text{ in } F \text{ and } \varphi(d(t)) = \varphi(t).$$

Sketch of Proof. The idea is to stuff the input with the dummy symbol d of rank 1, and to transform ε-rules of M into rules of M'' which read d; then φ is the identity on F and erases d. Formally, R''
contains all the read rules of M
contains, for each ε-rule $q(v, E(x_1,\ldots,x_s)) \to q'(v, \pi')$ of M, the read rule: $q(d(v), E(x_1,\ldots,x_s)) \to d(q'(v, \pi'))$.
The proof is left to the reader. $\square$

Lemma 3. *Let M'' be associated as in lemma* 2 *to the automaton M' constructed in lemma* 1. *Then $T(M') = \psi^{-1}(T(M''))$ where ψ is the linear homomorphism $A(F) \to A(F \cup \{d\})$ defined by: $\psi(f(t_1,\ldots,t_r)) = d^k(f(\psi(t_1),\ldots,\psi(t_r)))$ (d^k denotes the composition of d with itself k times).*

Sketch of Proof. By the construction of M' and M'' one can easily see that any tree accepted by M'' is of the form $\psi(t)$, for some t accepted by M'. $\square$

Now, $T(M'')$ is Greibach by proposition 5; $L = T(M') = \psi^{-1}(T(M''))$ is thus also Greibach, since (by theorem 24 of [4], Greibach languages are closed under inverse linear homomorphisms. $\square$

Corollary 2. *Every context-free tree language is the image of a Greibach language by a linear alphabetic homomorphism whose only erasing rules are monadic (monadic nonstrictness in Leguy's terminology* [21]).

This is simply a restatement of Lemma 2.

Let (Q) RT denote the class of (quasi)real-time languages, (D)RT denote the class of (deterministic) real-time languages, G(resp. GE) denote the class of languages which can be generated by a Greibach (resp. extended Greibach) grammar: an extended Greibach grammar is a grammar which would be Greibach but for erasing rules $E(x_1,\ldots,x_s)\to x_i$ which are allowed. Extended Greibach grammars are called Greibach in [5]. Let DG denote the class of languages generated by deterministic grammars. Let CFT denote the class of context-free tree languages, PDTA denote the class of languages accepted by PDTA's and PDTA_{pop} denote the class of languages accepted by PDTA's whose only ε-rules are "pop" rules: $q(v, E(x_1,\ldots,x_s))\to q(v', x_i)$. Then the following theorem summarizes the main results of this section:

Theorem 2

(i) $\text{D} = \text{DRT} = \text{DG}$

ii) $\text{RT} = \text{QRT} = \text{G}$

(iii) $\text{GE} = \text{PDTA}_{\text{pop}}$

(iv) $\text{D} \subsetneq \text{RT} \subsetneq \text{PDTA}_{\text{pop}} \subsetneq \text{PDTA} = \text{CFT}$

Proof. (i) and (ii) follow from propositions 5, 6, 7; (iii) is an easy consequence of the construction given in theorem 1; the strictness of the inclusions in (iv) is well-known, e.g. $L = \{f(a,a), f(b,b)\} \in \text{RT} - \text{D}$, $L' = \{g(d^n a, d^n a)/n \in N\} \in \text{CF} - \text{GE}$ [23], and $\text{G} \neq \text{GE}$ follows from [21]. □

5. Restricted PDTA's and Indexed Grammars

We will now give a more operational characterization of context-free tree languages by simplifying our PDTA's: the pushdown is a usual string instead of a tree (but it never gets emptied). Formally:

Definition 4. A *restricted pushdown tree automaton* (RPDTA) is a PDTA (Q, F, Π, q_0, Z_0, R) where $\Pi = \{Z_0\} \cup \Pi_1$, i.e. but for the start symbol Z_0 of rank 0, the pushdown alphabet contains only rank 1 symbols. The allowable rules are indicated below.

Notation. The pushdown store will thus always be of the form $H_1(H_2(\ldots(H_n(Z_0))\ldots))$: omitting the parentheses we will denote it by $w = H_1 H_2 \ldots H_n Z_0 \in \Pi_1^* Z_0$: i.e. composition is denoted by concatenation and $A(\Pi)$ is identified with the subset $\Pi_1^* Z_0$ of the gree-monoid over $\Pi_1 \cup \{Z_0\}$. Note that, because of the symbol Z_0, the store never is empty.

We allow the following types of rules for a RPDTA:

(i) $q(f(v_1,\ldots,v_r), Ex)\to f(q_1(v_1,\pi_1 x),\ldots,q_r(v_r,\pi_r x))$

(ii) $q(v, Ex)\to q'(v,\pi' x)$

(iii) $q(f(v_1,\ldots,v_r), x)\to f(q_1(v_1,\pi_1 x),\ldots,q_r(v_r,\pi_r x))$

(iv) $q(v, x)\to q'(v,\pi' x)$

with $f \in F_r$, $E \in \Pi_1$, $\pi' \in \Pi_1^*$, for $i = 1,\ldots,r$, $\pi_i \in \Pi_1^*$ and x a variable ranging over $\Pi_1^* Z_0$.

Note that (i) and (ii) are more restricted than those of arbitrary PDTA with $\Pi = \{Z_0\} \cup \Pi_1$.

In the notations of [11] an RPDTA can be viewed as a semi-thue system with variables $\mathscr{S} = (\mathscr{A}, \mathscr{X}, \mathscr{D}, \mathscr{R})$ with

$\mathscr{A} = F \cup \Pi \cup Q \cup \{(,), c\}$, where c is the comma.

$\mathscr{X} = V \cup X$, where $V = \{v, v', v_1, v_2, \ldots\}$ and $X = \{x\}$

variables in V ranging over $A(F)$, i.e. $\mathscr{D}(v) = A(F)$ for each v in V, and variables in X ranging over $\Pi_1^* Z_0$, i.e. $\mathscr{D}(x) = \Pi_1^* Z_0$

$\mathscr{R} = R$

The computation relation $\vdash^*$ of the RPDTA then coincides with the derivation relation $\overset{*}{\underset{\mathscr{S}}{\Rightarrow}}$ of the semi-thue system $\mathscr{S}$.

We can now state the main theorem of this section:

Theorem 3. *Any language accepted by a* PDTA *can be accepted by a restricted* PDTA.

To prove this theorem, one would expect to simply use the fact that, if L is context-free, then the set of branches of L is a context-free language, hence is recognized by a pushdown automaton. Then one could construct the corresponding PDTA (by "glueing" together productions of the pushdown automaton). Unfortunately, this PDTA will recognize a context-free tree language L' which is usually strictly greater than L (except when L is "branch closed" or "representable" [8, 27]).

One might then hope to derive this theorem from Gallier's result [16] by some trick: e.g. add a binary base symbol + (representing set-theoretic union) to eventually substitute ε-rules for all rules concerning the symbol +. The previous remark also shows that all such hopes are futile. We thus have to find a direct construction. It nevertheless will be strongly inspired by Gallier's construction.

We first give the idea of the construction. We have to show that every context-free language is a $T(M)$ for some restricted PDTA M. We can no more have a single state PDTA where the whole tree which remains to be derived is stored in the pushdown store. Hence we shall use both the state and the pushdown store to code (or "remember") the derivations which remain to be done: i.e., the state remembers at which occurrence we are currently located in the right-hand side trees at the present moment of the derivation, and the pushdown store remembers which occurrences of variable function symbols still have to be derived. The state and the pushdown store are then used interactively to reconstruct the derivation of a tree.

Proof of Theorem 3. Let $\mathscr{G} = (F, \Phi, P, G_0)$ be a context-free tree grammar with productions:

$$G_i(x_1,\ldots,x_{r_i}) \to \{t_i^j,\ j = 1,\ldots,n_i\} \qquad i = 0,\ldots,n.$$

Then let $M = (Q, F, \Pi, q_0, G_0, R)$ be defined by: $Q = \{q_0\} \cup \{(i, j, o)/i = 0,\ldots,n, j = 1,\ldots,n_i$ and o is an occurrence in $t_i^j\}$.

The input alphabet F is the terminal alphabet of $\mathscr{G}$, $\Pi = \{(i, j, o)/i = 0,\ldots,n,$ $j = 1,\ldots,n_i$ and o is an occurrence of a $G_k \in \Phi$ in $t_i^j\} \cup \{G_0\}$, and the rules are defined by:

(1) initializations by ε-rules of type (iv):

$$q_0(v, x) \to (0, j, \varepsilon)(v, x) \quad \text{for } j = 1,\ldots,n_0.$$

(2) to each occurrence o of a base function symbol f in a t_i^j (i.e. $t_i^j(o) = f$) corresponds a type (iii) read move:

$$(i, j, o)(f(v_1,\ldots,v_r), x) \to f((i, j, o1)(v_1, x),\ldots,(i, j, or)(v_r, x))$$

Intuitively, reading f in a state corresponding to the position o in the tree t_i^j, we have to go down in the input tree without changing the store.

Notice that in the special case where f has rank 0, we get accepting rules which erase the stack.

(3) to each occurrence o of a variable function symbol G_k in a t_i^j (i.e. $t_i^j(o) = G_k$), correspond push moves where we store G_k in the pushdown, thus remembering the recursive call which shall be done later, and reposition ourselves in a state which corresponds to beginning the derivation of G_k (i.e. at the roots of the right-hand sides $t_k^{j'}$ corresponding to G_k); i.e. we get the type (iv) ε-rule:

$$(i, j, o)(v, x) \to (k, j', \varepsilon)(v, (i, j, o)x),$$

for $j' = 1,\ldots,n_k$, whenever $t_i^j(o) = G_k \in \Phi$.

(4) to each occurrence of a variable x_m indicating that the current recursive call has been completed, corresponds a pop move, i.e. going to the next recursive call and repositioning (by means of the state) to the x_m argument of each occurrence of the popped symbol in the t_i^j's; i.e. we get the type (ii) ε-rule:

$$(i, j, o)(v, (i', j', o')x) \to (i', j', o'm)(v, x)$$

for any (i, j, o), (i', j', o') such that $t_i^j(o) = x_m \in X_{r_i}$ and $t_{i'}^{j'}(o') = G_i \in \Phi$ and $X_{r_i} = \{x_1,\ldots,x_{r(G_i)}\}$.

Then, $L(\mathscr{G}) = T(M)$ follows from the lemma:

Lemma. *Let $G_i(x_1,\ldots,x_{r_i}) \to t_i^j$ be a rule of $\mathscr{G}$. Let $t_i'^j$ be the prefix of t_i^j such that:*

$t_i'^j \in A(F \cup \Phi, W_n)$ with $W_n = \{w_1,\ldots,w_n\}$ disjoint from $X_{r_i} = \{x_1,\ldots,x_{r_i}\}$

no two leaves of $t_i'^j$ are labeled by the same variable w_k

$t_i^j = t_i'^j(x_{i_1}/w_1,\ldots,x_{i_n}/w_n)$, $x_{i_k} \in X_{r_i}$ for $k = 1,\ldots,n$.

Let t' be a subtree of $t_i'^j$ at occurrence o. Then $t' \overset{*}{\Rightarrow} t'' \in A(F, W_n)$ by an OI *derivation* iff: $(i, j, o)(t'', \psi) \overset{*}{\vdash} t''(((i, j, o_1)(w_1, \psi))/w_1,\ldots,((i, j, o_n)(w_n, \psi))/w_n)$ *is a computation sequence of M, where ψ is an arbitrary pushdown in $A(\Pi)$, and $t_i'^j(o_k) = w_k$, for $k = 1,\ldots,n$.*

The proof proceeds by induction on the length of the OI derivation and of the computation sequence. □

Let now t be in $L(\mathcal{G})$, then $G_0 \to t_0^j \overset{*}{\Rightarrow} t$ is a derivation sequence of t and applying the lemma with $t_i^j = t_0^j$, $\psi = G_0$ and $t' = t_0^j$ (note that $n = 0$) shows that $t \in T(M)$. Conversely, any accepting computation sequence of t in $T(M)$ has to start with some move $q_0(t, G_0) \vdash (0, j, \varepsilon)(t, G_0)$ and again we apply the lemma with $t_i^j = t_0^j$. □

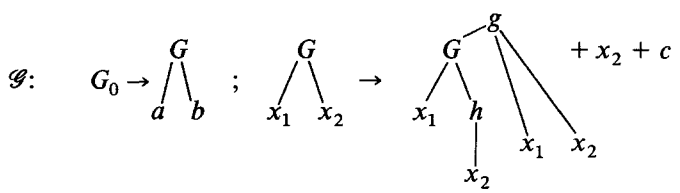
Remark 5. Note that:

1. The above construction is a generalization of LL parsing to a "parallel LL parsing" of trees: we read the tree in a topdown (or left to right) manner and construct an OI (or leftmost) derivation, by the usual method of stacking return addresses of recursive calls on the pushdown store. An alternate approach would be the one corresponding to the bottom-up TPDA's of [29], leading to a "parallel LR parsing": the storage of TPDA's consisting of set of trees whose roots are simultaneously accessible, an analogue of theorem 3 would be very hard to find: it thus seems impossible to linearize the storage of TPDA's.

2. This construction can lead to a very simple parsing method for indexed languages (see proposition 9 and theorem 4 below). It does not use Fischer's macrogrammars [15], as is done in [29].

Let us apply the above construction to the following simple example:

Example 4. Let $\mathcal{G}$ be the following grammar:



Since $n = n_0 = 1$ and $n_1 = 3$, we shall omit the components i, j in the states and pushdown symbols and simplify the notations as follows:

$(0,1,\varepsilon)$ is denoted by ε^0

$(0,1,1)$ (resp. $(0,1,2)$) is denoted by 1^0 (resp. 2^0)

$(1,1,\varepsilon)$ (resp. $(1,2,\varepsilon)$, $(1,3,\varepsilon)$) is denoted by ε(resp. ε', ε'').

$(1,1,o)$ with $o \neq \varepsilon$ is denoted by o; thus, e.g. ε'' corresponds to the root of t_1^3 which is labeled by c, 12 corresponds to occurrence 12 in t_1^1 labeled by h (i.e. $t_1^1(12) = h$). The pushdown alphabet is then $\Pi = \{\varepsilon^0, 1, G_0\}$. The rules of M are defined by:

initialization: $q_0(v, x) \to \varepsilon^0(v, x)$

occurrences of base function symbols:

$1^0(a, x) \to a$; $2^0(b, x) \to b$ and $\varepsilon''(c, x) \to c$

$12(h(v), x) \to h(121(v, x))$

$\varepsilon(g(v_1, v_2, v_3), x) \to g(1(v_1, x), 2(v_2, x), 3(v_3, x))$

occurrences of function variables:
$\varepsilon^0(v,x) \to \varepsilon(v,\varepsilon^0 x) + \varepsilon'(v,\varepsilon^0 x) + \varepsilon''(v,\varepsilon^0 x)$
$1(v,x) \to \varepsilon(v,1x) + \varepsilon'(v,1x) + \varepsilon''(v,1x)$
occurrences of variables:
occurrences of x_1:
for $q \in \{11,2\}$ $q(v,1x) \to 11(v,x)$ and
$q(v,\varepsilon^0 x) \to 1^0(v,x)$
occurrences of x_2:
for $q \in \{121,3,\varepsilon'\}$ $q(v,1x) \to 12(v,x)$
$q(v,\varepsilon^0 x) \to 2^0(v,x)$.

Remark 7. Alternatively, we might modify the definition of a RPDTA by letting $\Pi = \Pi_1$: we would then have to allow for rules with a possibly empty store (e.g. accepting rules on leaves, initializing rules,...). The bottom of pushdown G_0 symbol's sole use is in fact to ensure that the pushdown shall never get empty.

It is well known that context-free tree languages are the sets of derivation trees of indexed languages [12, 15]. Now, we shall see that PDTA's can easily be viewed as indexed grammars and thus provide, to our belief, some more insight into the parsing of indexed languages [2]. Essentially, a RPDTA can be viewed as a slight generalization of an indexed grammar, as will be shown by the construction below, due to J. Engelfriet.

Recall first the definition of an indexed grammar from [1].

Definition 5. An *indexed grammar* is a 5-tuple $\mathcal{G} = (N, T, F, P, S)$ where:

a) T(resp. N) is a finite alphabet of terminal (resp. nonterminal) symbols

b) S, the axiom, is a distinguished symbol in N

c) F is a finite set of *indexes*;

each f in F is a finite set of index productions of the index f of the form $A \to \omega$, where A is in N and ω in $(N \cup T)^*$.

d) P is a finite set of ordinary productions of the form $A \to \alpha$, where A is in N and α in $(NF^* \cup T)^*$.

The immediate rewriting relation $\underset{\mathcal{G}}{\Rightarrow}$ according to $\mathcal{G}$ is defined by: $w \underset{\mathcal{G}}{\Rightarrow} w'$ iff for w, w', w_1, w_2 in $(NF^* \cup T)^*$, x_i in $N \cup T$ and $\varphi, \varphi_i, \varphi_i'$ in F^* for $i = 1,\ldots,k$,

1. Either $w = w_1 A \varphi w_2$, $A \to x_1\varphi_1 \ldots x_k\varphi_k$ is an ordinary production in P and $w' = w_1 x_1 \varphi_1' \ldots x_k \varphi_k' w_2$, where for $i = 1,\ldots,k$ $\varphi_i' = \varphi_i\varphi$ if x_i is in N and $\varphi_i' = \varepsilon$ if x_i is in T.

2. Or $w = w_1 A f \varphi w_2$, $A \to x_1 \ldots x_k$ is in index f and $w' = w_1 x_1 \varphi_1 \ldots x_k \varphi_k w_2$, where for $i = 1,\ldots,k$, $\varphi_i = \varphi$ if x_i is in N and $\varphi_i = \varepsilon$ if x_i is in T.

As usual, $\overset{*}{\underset{\mathcal{G}}{\Rightarrow}}$ is the reflexive and transitive closure of $\underset{\mathcal{G}}{\Rightarrow}$, and $L(\mathcal{G}) = \{w \in T^*/S \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} w\}$.

Note that we may suppose without loss of generality that the only production rules in which terminals occur in right hand sides are of the form $A \to a$, or $A \to a$ in some index f. For each occurrence of a terminal a in a right-hand side w which is not of the above form, introduce a new nonterminal A', substitute A' for a in w and add the production rule $A' \to a$ in P.

From now on, we will suppose that indexed grammars are of this restricted form, namely that in Definition 5 above, in case c) ω is in $N^* \cup T$ and in case d) α is in $(NF^*)^* \cup T$.

Definition 6. An *indexed top-down tree automaton* (ITA) is an RPDTA with rules of the form:

(i′) $q(f(v_1,\ldots,v_r), Ex) \to f(q_1(v_1, x),\ldots,q_r(v_r, x))$

(ii′) $q(v, Ex) \to q'(v, x)$

(iii) $q(f(v_1,\ldots,v_r), x) \to f(q_1(v_1, \pi_1 x),\ldots,q_r(v_r, \pi_r x))$

(iv) $q(v, x) \to q'(v, \pi' x)$

where f is in F_r, π' and π_i, for $i = 1,\ldots,r$, are in Π_1^*, and x ranges over $\Pi_1^* Z_0$; namely, $\pi' = \pi_1 = \cdots = \pi_r = \varepsilon$ in type (i) and (ii) rules.

Lemma 4. *Any* RPDTA *is equivalent to an* ITA.

Proof. For each type (i) rule

m: $q(f(v_1,\ldots,v_r), Ex) \to f(q_1(v_1, \pi_1 x),\ldots,q_r(v_r, \pi_r x))$, introduce r new states $q_1^{\pi_1},\ldots,q_r^{\pi_r}$ and replace rule m by the following $r+1$ rules:

$q(f(v_1,\ldots,v_r), Ex) \to f(q_1^{\pi_1}(v_1, x),\ldots,q_r^{\pi_r}(v_r, x))$ of type (i′), and, for $i = 1,\ldots,r$, $q_i^{\pi_i}(v, x) \to q_i(v, \pi_i x)$, of type (iv). Similarly decompose each type (ii) rule:

$q(v, Ex) \to q'(v, \pi' x)$ into the two rules

$q(v, Ex) \to q'_{\pi'}(v, x)$ of type (ii′) and

$q'_{\pi'}(v, x) \to q'(v, \pi' x)$ of type (iv). □

Proposition 8. *To each* ITA *corresponds an indexed grammar generating its yield.*

Proof. Recall first (cf. proposition 4) that, by dropping all first arguments of states, we can view a PDTA as a rewriting system generating the tree language accepted by the PDTA.

Now, let $M = (Q, F, \Pi, q_0, Z_0, R)$ be an ITA. First dropping all first arguments of states and then applying the yield homomorphism to the right hand sides in $A(F, Q \times \Pi^*)$, but not to the pushdowns, we obtain a set of rules R'' with the following types of rules:

(i″) $q(Ex) \to q_1(x)\ldots q_r(x)$ $\quad r \geq 1$

or $q(Ex) \to a$ $\quad a \in F_0$

(iii″) $q(x) \to q_1(\pi_1 x)\ldots q_r(\pi_r x)$ $\quad r \geq 1$

or $q(x) \to a$ $\quad a \in F_0$

We may omit rules (ii″) and (iv′) corresponding to (ii′) and (iv) because they lead to special cases of (i″) and (iii″) with $r = 1$.

Now x's in the above rules can be viewed as variables ranging over the set F^* of words on indexes of some indexed grammar $\mathcal{G}$ where $F = \Pi$. Namely, rules (i″) correspond to index productions, and mean that index E contains the production $q \to q_1\ldots q_r$ or $q \to a$, and (iii″) correspond to ordinary productions, and mean that $q \to q_1\pi_1\ldots q_r\pi_r$ or $q \to a$ are ordinary productions of $\mathcal{G}$ (in Aho's notations). Let $N = Q \cup \{S\}$, $T = F_0$, $F = \Pi$, P be the set of productions:

(i) for each E in Π, q in Q and type (i″) rules of R'', index E contains the index productions $q \to q_1\ldots q_r$ and $q \to a$, and nothing else.

(iii) for each q in Q and type (iii″) rules of R'', P contains ordinary productions $q \to q_1\pi_1\ldots q_r\pi_r$ and $q \to a$

(o) initialization: letting S be the axion, P contains rule $S \to q_0 Z_0$. Each index E in Π is identified with the corresponding set of productions (i).

Then, the above defined indexed grammar $\mathcal{G}$ is such that $L(\mathcal{G}) = \text{yield}(T(M))$. □

Theorem 4 [15, 12]. *The class of indexed languages coincides with the class of yields of context-free tree languages.*

Proof. Proposition 8 shows that the yield of any context-free tree language is indexed. Conversely, the construction in proposition 8 immediately shows how to construct an ITA M accepting the language generated by an indexed grammar. If $\mathcal{G} = (N, T, F, P, S)$ then $M = (N, T \cup \bigcup_{r \geq 1} \{c_r\}, F \cup \{Z_0\}, S, Z_0, R)$, where for each $r \geq 1$, c_r is a rank r symbol intended to represent the r-ary concatenation, and R is deduced from P by introducting explicitly the concatenation operators.

Formally, for each index production $A \to A_1 \ldots A_r$ or $A \to a$ in index E in F, R contains type (i') rules:

$$A(c_r(v_1, \ldots, v_r), Ex) \to c_r(A_1(v_1, x), \ldots, A_r(v_r, x)) \text{ or } A(a, Ex) \to a.$$

For each ordinary production $A \to A_1\varphi_1 \ldots A_r\varphi_r$, with $\varphi_i \in F^*$ for $i = 1, \ldots, r$, (resp. $A \to a$ production in P), R contains type (iii) rules: $A(c_r(v_1, \ldots, v_r), x) \to c_r(A_1(v_1, \varphi_1 x), \ldots, A_r(v_r, \varphi_r x))$ (resp. $A(a, x) \to a$) where x ranges over F^*Z_0.

Then yield$(T(M)) = L(G)$, except possibly for the empty string which may belong to $L(G)$ but not to yield$(T(M))$. □

The above construction, besides providing an alternate proof of theorem 4, has the advantage of giving an explicit very simple transformation (basically a homomorphism) for going from a restricted pushdown automaton to the corresponding indexed grammar.

Acknowledgments

It is a pleasure to thank J. Engelfriet whose insightful suggestions tremendously helped in improving this paper, A. Arnold, W. Damm and J. Gallier for helpful comments and/or discussions, and Mme A. Dupont for her patient typing. Thanks also to the referees for their careful reading and constructive critics.

References

1. A. V. Aho, Indexed grammars: An extension of the context-free case, *JACM* 15, 647–671 (1968).
2. A. V. Aho, Nested stack automata, *JACM* 16, 383–406 (1969).
3. A. Arnold, M. Dauchet, Transductions de forêts reconnaissables monadiques. Forêts corégulières, RAIRO Inf. Théor. 10, 5–28 (1976).
4. A. Arnold, B. Leguy, Une propriété des forêts algébriques de Greibach, Information & Control, Vol. 46, 108–134 (1980).
5. A. Arnold, M. Nivat, Formal computations of nondeterministic program schemes, *MST* 13, 219–236 (1980).
6. J. Berstel, *Transductions and Context-Free Languages*, Teubner, Stuttgart (1979).

7. G. Boudol, Langages algébriques d'arbres, LITP report n° 81–2.
8. B. Courcelle, On jump deterministic pushdown automata, *MST* 11, 87–109 (1977).
9. W. Damm, Personal communication.
10. P. Downey, Formal languages and recursion schemes, Ph.D. Harvard (1974).
11. J. Engelfriet, Bottom-up and top-down tree transformations. A comparison, *MST* 9, 198–231 (1975).
12. J. Engelfriet, Some open questions and recent results on tree transducers and tree languages, in *Formal Languages: Perspectives and Open Problems*, Academic Press, London (1980), 241–286.
13. J. Engelfriet, E. M. Schmidt, IO and OI, *JCSS* 15, 328–353, (1977) and *JCSS* 16, 67–99 (1978).
14. J. Engelfriet, G. Slutzki, Extended macrogrammars and stack controlled machines, Memo n° 379, T. H. Twente, (1982).
15. M. J. Fischer, Grammars with macro-like productions, 9th *SWAT*, 131–142 (1968).
16. J. H. Gallier, DPDA's in "Atomic normal form" and applications to equivalence problems, *TCS* 14, 155–186 (1981).
17. S. Gorn, Explicit definitions and linguistic dominoes, in J. Hart, S. Takasu, Eds, *Systems and Computer Science* (1965).
18. I. Guessarian, *Algebraic semantics*, Lect. Notes in Comp. Sc. n° 99, Springer-Verlag, Berlin (1981).
19. M. Harrison, *Introduction to Formal Languages Theory*, Addison-Wesley, London, (1978).
20. J. E. Hopcroft, A. J. Korenjak, Simple deterministic languages, 7th Symp. Switching and Automata Theory, Berkeley, 36–46 (1966).
21. B. Leguy, Réductions, transformations et classification des grammaires algébriques d'arbres, Thèse 3ème Cycle, Univ. Lille 1 (1980).
22. M. Nivat, On the interpretation of recursive polyadic program schemes, Symp. Mat. 15, Rome, 255–281 (1975).
23. N. Polian, Adhérences et centres de langages d'arbres, Thèse de 3ème cycle, Poitiers (1981).
24. M. Rabin, Automata on infinite objects and Church's problem, CBSM regional Conf. series in Math. n° 13, AMS (1969).
25. W. C. Rounds, Mappings and grammars on trees, *MST* 4, 257–287 (1970).
26. W. C. Rounds, Tree oriented proofs of some theorems on context-free and indexed languages, 2nd Symposium on Theory of Computing, 109–116 (1970).
27. A. Saoudi, Forêts infinitaires reconnaissables, Thèse de 3ème cycle, Université Paris 7 (1982).
28. K. Schimpf, A parsing method for context-free tree languages, Ph.D. University of Pennsylvania (1982).
29. K. Schimpf, J. Gallier, Parsing tree languages using bottom-up tree automata with tree pushdown stores, Extended abstract, University of Pennsylvania (1981), to appear.
30. J. M. Steyaert, Lemmes d'itération pour les familles d'arbres, Actes du Séminaire d'Inf. Théor. 1977–1978, LITP Report, Paris (1979).
31. J. W. Thatcher, Tree automata: An informal survey, in *Currents in Theory of Comp.*, Aho (ed.), Prentice-Hall, London (1973).
32. J. W. Thatcher, J. B. Wright, Generalized finite automata theory with an application to a decision problem of second order logic, *MST* 2, 57–81 (1968).

Received February 5, 1982, and in revised form May 28, 1982, January 12, 1983, and April 15, 1983.