Theoretical Computer Science 41 (1985) 305–318
North-Holland

ALTERNATING TREE AUTOMATA*

Giora SLUTZKI
Department of Computer Science, Iowa State University, Ames, IA 50011, U.S.A.

Communicated by M. Nivat
Received February 1983
Revised September 1985

1. Introduction

In [2, 9], the concept of alternation has been introduced as a natural extension of the concept of nondeterminism. Intuitively, once we can think of a nondeterministic machine as an automaton all of whose configurations are existential, it is natural to generalize by distinguishing between existential and universal configurations. A configuration α is universal if all computations starting from α are accepting computations. The concept of existential configurations is the standard one. The effect of alternation was studied in the context of computational complexity, finite automata, and (one-way and two-way) pushdown (and stack) automata [2, 3, 9, 14, 17]. Some applications of alternation have been considered in algebra [1, 10], in analysis of propositional dynamic logic [8] and combinatorial games [12, 18].

In this paper we discuss the effect of alternation on several varieties of tree automata. With respect to each class of automata we distinguish the following four subclasses:

(i) *alternating automata*—allowing the full power of alternation,
(ii) *nondeterministic automata*—all configurations are existential,
(iii) *universal automata*—all configurations are universal,
(iv) *deterministic automata*—the transition function is a partial function.

An a priori inclusion relationship between these four features is shown in Fig. 1. For some types of automata, nondeterminism and universality turn out to be incomparable, while for others the whole diagram 'collapses into determinism'.

The paper consists of seven sections including the introduction. In the next section we recall some standard notation from tree language theory and in Section 3 we discuss alternating (one-way) top-down automata. For these automata we prove the equivalence of universality and determinism. Alternation and nondeterminism are also equivalent, but this will be proved in Section 5. In Section 4 we

* This work was supported in part by the University of Kansas General Research Allocation under Grant No. 413-3232-20-0038.

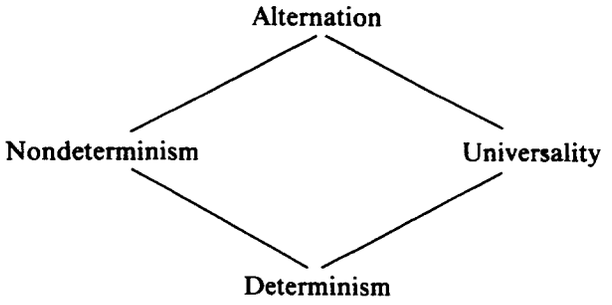


Fig. 1.

consider alternating two-way finite tree automata. In this case, universality and nondeterminism are incomparable and the diagram of Fig. 1 is correct. In Sections 5 and 6 we discuss two types of alternating two-way pushdown tree automata. In Section 5 we study the synchronized type [7, 13] and in Section 6 we study the backtracking type [11]. In both cases we show that alternation is equivalent to determinism. The motivation for alternating two-way push-down tree automata comes from their connection to generalized syntax-directed translation for the synchronized type [7], and to the attribute grammars for the backtracking type [11]. In Section 7 we summarize our results.

2. Preliminaries

An alphabet Σ is *ranked* if $\Sigma = \bigcup_k \Sigma_k$ where each Σ_k is a finite set and, only for a finite number of k's, $\Sigma_k \neq \emptyset$. Elements of Σ_k are said to be of rank k. Given a ranked alphabet Σ, the set of trees over Σ, denoted T_Σ, can be considered as a language over the alphabet $\Sigma \cup \{(\,,\,)\}$ recursively defined as follows:

(i) $\Sigma_0 \subseteq T_\Sigma$,

(ii) if $k \geq 1$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k \in T_\Sigma$, then $\sigma(t_1 \ldots t_k) \in T_\Sigma$.

Let Σ be a ranked alphabet and let S be a set of symbols or trees. The set of trees over Σ *indexed by* S, denoted $T_\Sigma[S]$, is recursively defined as follows:

(i) $S \cup \Sigma_0 \subseteq T_\Sigma[S]$,

(ii) if $k \geq 1$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k \in T_\Sigma[S]$, then $\sigma(t_1 \ldots t_k) \in T_\Sigma[S]$.

Let $X = \{x_1, x_2, \ldots\}$ be an infinite set of variables. These are used in the production rules of top-down tree automata.

For any set S, $\mathcal{P}(S)$ ($\mathcal{P}_{\text{fin}}(S)$) denotes the set of all (finite) subsets of S.

3. Alternating one-way top-down tree automata

Parallel (one-way) top-down tree automata, nondeterministic and deterministic, are well known [4, 5, 6, 15, 16, 19, 20]. The relations between these are also well known: the nondeterministic automata characterize a class of tree languages called RECOG and they are more powerful than the deterministic automata. Here we

generalize by introducing alternation (and universality). It turns out that alternating and nondeterministic top-down tree automata are equipotent; the same holds for universal and deterministic top-down automata.

We proceed to the definitions.

Definition 3.1. An *alternating (one-way) top-down tree automaton* (atta) is a construct $M=(Q, U, q_0, \Sigma, R)$ where Q is a finite nonempty set of states, $U \subseteq Q$ is the set of *universal states* (states in $Q-U$ are called *existential states*), $q_0 \in Q$ is the initial state, Σ is a ranked input alphabet, and R is a finite set of rules of the form $q(\sigma(x_1 \dots x_k)) \to \sigma(q_1(x_1) \dots q_k(x_k))$, where $k \geq 0$, $\sigma \in \Sigma_k$, and $q, q_1, \dots, q_k \in Q$.

An *instantaneous description* (ID) of M on a tree t in T_Σ is a tree in the set $T_\Sigma[Q(T_\Sigma)]$, where $Q(T_\Sigma)$ is the set of trees $\{q(t) \mid q \in Q, t \in T_\Sigma\}$ (here we view states as having rank 1). $q_0(t)$ is the *initial* ID of M on t, and trees in T_Σ are the *accepting* ID's. For two ID's s and r we write $s \vdash_M r$ if there is a rule $q(\sigma(x_1 \dots x_k)) \to \sigma(q_1(x_1) \dots q_k(x_k))$ in R such that r is obtained from s by replacing a subtree of s of the form $q(\sigma(t_1 \dots t_k))$ for certain $t_1, \dots, t_k \in T_\Sigma$ by $\sigma(q_1(t_1) \dots q_k(t_k))$. Given $\vdash_M$, $\vdash_M^*$ is the reflexive, transitive closure of $\vdash_M$.

A *computation tree of M on t* is a finite, nonempty tree labeled by ID's of M (on t) and satisfying the following properties.

(i) The root of the tree is labeled by the initial ID of M on t: $q_0(t)$.

(ii) Let n be an internal node of the tree labeled by an ID s and let $q(\sigma(t_1 \dots t_k))$ be a subtree of s (at node π) for some $\sigma \in \Sigma_k$ and $t_1, \dots, t_k \in T_\Sigma$. Let $r_1, \dots, r_m$ be all the rules in R that have $q(\sigma(x_1 \dots x_k))$ as a left-hand side and denote by s_i the ID obtained from s by application of rule r_i at node π of s.

(a) If q is a universal state (in U), then n will have m sons $n_1, \dots, n_m$ labeled respectively by the ID's $s_1, \dots, s_m$.

(b) If q is an existential state (in $Q-U$), then n will have a single son n' labeled by one of the ID's s_i, $1 \leq i \leq m$.

An *accepting computation tree of M* on t is a computation tree of M on t whose leaves are all labeled by accepting ID's. M accepts the tree t if there exists an accepting computation tree of M on t. The tree language defined by M is $L(M)=\{t \in T_\Sigma \mid M \text{ accepts } t\}$. An atta is *universal* (utta) if $U=Q$ and it is *nondeterministic* (ntta) if $U=\emptyset$. An atta is *deterministic* (dtta) if for all $k \geq 0$, $\sigma \in \Sigma_k$, and $q \in Q$, there is at most one rule with left-hand side $q(\sigma(x_1 \dots x_k))$. Names for the various classes of automata defined above are obtained by capitalizing all the letters, e.g., the class of all universal top-down tree automata is denoted by UTTA. The names of the families of tree languages defined by these classes of automata are obtained by changing the last letter "A" of the automata-class name to "L"; for example, ATTL is the family of tree languages recognizable by automata in ATTA. We shall keep those notational conventions throughout the paper.

Definition 3.2. The class of tree languages NTLL will be denoted by RECOG. A tree language L in RECOG is said to be *recognizable*.

We now show that the features of universality and determinism are equivalent for ATTA's.

Theorem 3.3. DTTL = UTTL.

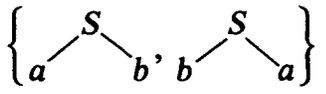
Proof. Clearly, it suffices to prove UTTL $\subseteq$ DTTL. We use a simple subset construction as follows. Let $M = (Q, Q, q_0, \Sigma, R)$ be a utta; the corresponding dtta is $N = (\mathcal{P}(Q), \mathcal{P}(Q), \{q_0\}, \Sigma, \hat{R})$ where the new rules are derived from the old ones as follows:

(i) For $Q_0 \subseteq Q$ and $\sigma \in \Sigma_k$, $Q_0(\sigma(x_1 \dots x_k)) \to \sigma(Q_1(x_1) \dots Q_k(x_k))$ is in $\hat{R}$, where the Q_i $(i \geq 1)$ are subsets of Q with $q_i \in Q_i$ iff there is a $q \in Q_0$ such that $q(\sigma(x_1 \dots x_k)) \to \sigma(\dots q_i(x_i) \dots)$ is a rule in R.

(ii) For $Q_0 \subseteq Q$ and $\sigma \in \Sigma_0$, $Q_0(\sigma) \to \sigma$ is in $\hat{R}$ iff for every $q \in Q_0$ the rule $q(\sigma) \to \sigma$ is in R.

It is easy to see that $L(M) = L(N)$. □

The recognizable tree language



is known not to be in DTTL. Thus, for ATTL's we obtain the inclusion diagram of Fig. 2. A broken line means an inclusion not yet proved to be either proper or equality. In the case of ATTL we shall prove equality in Section 5.

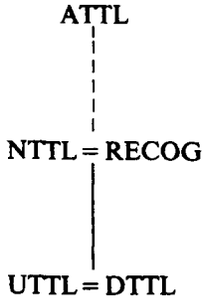


Fig. 2.

Remark. Although it is not clear how to define alternating bottom-up automata, we may define the universal class in a natural way. By an easy subset construction we can show that universal bottom-up tree automata are equivalent to the deterministic version and hence exactly recognize RECOG.

4. Alternating two-way finite tree automata

Nondeterministic and universal two-way finite tree automata have been introduced and studied in [11, 13]. In this section we generalize these concepts by introducing alternation.

Definition 4.1. An *alternating two-way finite tree automaton* (2ata) is a construct $M=(Q, U, \Sigma, \delta, q_0, F)$, where Q is a finite nonempty set of states, $U \subseteq Q$ is the set of *universal states* (states in $Q-U$ are called *existential states*), Σ is a ranked input alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, and δ is the transition function $\delta: Q \times \Sigma \to \mathcal{P}(Q \times D)$, where $D=\{-1, 0, 1, 2, \ldots, m)$ with m being the maximal i such that $\Sigma_i \neq \emptyset$. δ satisfies the condition that, for every $q \in Q$ and every $a \in \Sigma_k$ $(0 \leq k \leq m)$, $(p, i) \in \delta(q, a)$ implies $-1 \leq i \leq k$.

An *instantaneous description* (ID) of M on a tree t in T_Σ is a triple of the form (q, n, t) where $q \in Q$, $t \in T_\Sigma$ and n is a node of t or ω. A *universal* (*existential*) ID is an ID (q, n, t) with $q \in U$ $(q \in Q-U)$. An *accepting* (*rejecting*) ID is one in which $q \in F$ $(q \in Q-F)$ and $n=\omega$. The *initial* ID is (q_0, r, t) with r being the root of t.

We next define the computation relation $\vdash_M$ between ID's of M. If $I=(q, \omega, t)$, then there is no ID J for which $I \vdash_M J$. For $n \neq \omega$,

$$(q, n, t) \vdash_M (p, n', t)$$

if $(p, i) \in \delta(q, a)$ where a is a label of n and n' is given by the following self-explanatory code:

$$\begin{aligned} n' := \textbf{if } i=-1 \textbf{ then if } n = \text{root-of}(t) \textbf{ then } \omega \\ \textbf{else } \text{father}(n) \\ \textbf{else if } i=0 \textbf{ then } n \\ \textbf{else } i\text{th-son-of}(n). \end{aligned}$$

The reflexive, transitive closure of $\vdash_M$ is denoted by $\vdash_M^*$.

A *computation tree of M on t* is a nonempty (not necessarily finite) tree labeled by ID's of M (on t) and satisfying the following properties:

(i) The root of the tree is labeled by the initial ID of M on t.

(ii) If n is an internal node of the tree and label$[n]$ (the ID labeling the node n) is an existential ID, then n has a single son n' and its label must satisfy label$[n] \vdash_M$ label$[n']$.

(iii) If n is an internal node of the tree, label$[n]$ is a universal ID and $\{I \mid \text{label}[n] \vdash_M I\} = \{I_1, \ldots, I_k\}$, then n has k sons $n_1, \ldots, n_k$ such that, for each $1 \leq i \leq k$, label$[n] \vdash_M I_i =$ label$[n_i]$.

An *accepting* (*rejecting*) *computation tree* of M on t is a finite computation tree of M on t whose leaves are (not) all labeled by accepting ID's. The automaton M *accepts* the tree t if there exists an accepting computation tree of M on t. The tree language accepted by M is $L(M)=\{t \in T_\Sigma \mid M \text{ accepts } t\}$. A 2ata is a *universal* two-way tree automaton (2uta) if $U=Q$ and it is a *nondeterministic* two-way tree automaton (2nta) if $U=\emptyset$; these two classes of automata were defined in [11] and [13] respectively. Deterministic two-way tree automata (2dta) are obtained from 2ata's by requiring that the transition function is a partial function. Names for the various classes of automata defined above are obtained by capitalizing the letters, and the names of the families of tree languages characterized by these varieties of

tree automata are obtained by changing the last letter of automata-class name from "A" to "L", exactly as in Section 3.

Example 4.2. Let $\Sigma = \Sigma_0 \cup \Sigma_2$, where $\Sigma_0 = \{a, b\}$ and $\Sigma_2 = \{A\}$. Define a tree language $L = \{t \in T_\Sigma \mid \text{all leaves of } t \text{ are labeled by } a\}$.

(i) The 2uta $M = (\{q, d, p\}, \{q, d, p\}, \Sigma, \delta, q, \{p\})$, where $\delta(q, A) = \{(q, 1), (q, 2)\}$, $\delta(q, a) = \{(p, -1)\}$, $\delta(q, b) = \{(d, -1)\}$, $\delta(p, A) = \{(p, -1)\}$, and $\delta(d, A) = \{(d, -1)\}$ accepts exactly L.

(ii) The 2nta $N = (\{q, d, p\}, \emptyset, \Sigma, \delta, q, \{d\})$ accepts exactly $\bar{L}$, the complement of L.

In [13] it was shown that $\bar{L}$ is not in 2DTL and that L is not in 2NTL. In [11] it is shown that $\bar{L}$ is not in 2UTL. Since, by Example 4.2, L is in 2UTL and $\bar{L}$ is in 2NTL, it follows that 2NTL and 2UTL are incomparable and so the inclusion diagram of Fig. 3 is correct.

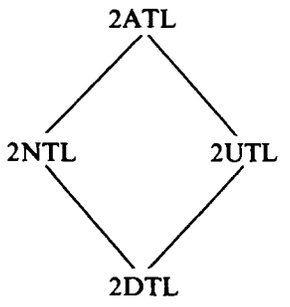


Fig. 3.

Lemma 4.3. 2UTL $\cup$ 2NTL *is properly included in* 2ATL.

Proof. Take $\Sigma = \{\wedge, \vee, \mathrm{T}, \mathrm{F}\}$ where $\Sigma_0 = \{\mathrm{T}, \mathrm{F}\}$ and $\Sigma_2 = \{\wedge, \vee\}$. Each tree in T_Σ represents a propositional formula with truth values (assuming T means 'true' and F means 'false') as atoms. The language $L_\mathrm{T} = \{t \in T_\Sigma \mid \text{value of } t \text{ is 'true'}\}$ is accepted by the following 2ata $M = (\{q, e, u, p, f\}, \{u\}, \Sigma, \delta, q, \{p\})$, where $\delta(q, \wedge) = \{(u, 0)\}$, $\delta(q, \vee) = \{(e, 0)\}$, $\delta(q, \mathrm{T}) = \{(p, -1)\}$, $\delta(q, F) = \{(f, -1)\}$, $\delta(u, \wedge) = \delta(e, \vee) = \{(q, 1), (q, 2)\}$, $\delta(p, \wedge) = \delta(p, \vee) = \{(p, -1)\}$, and $\delta(f, \wedge) = \delta(f, \vee) = \{(f, -1)\}$.

L_T can be shown not to be in 2NTL or 2UTL by exactly the same methods that were used to show that languages L and $\bar{L}$ are not in 2NTL and 2UTL (see respectively [13, Theorem 5.5] and [11, Lemma 3.4]). □

Now we prove the following theorem.

Theorem 4.4. ATTL $\subseteq$ 2ATL.

Proof. Let $M = (Q, U, q_0, \Sigma, R)$ be an atta. We construct a 2ata, $N = (Q \cup P \cup \{q_F\}, U \cup P, \Sigma, \delta, q_0, \{q_F\})$ as follows. Let $q(\sigma(x_1 \ldots x_k))$ be the left-hand side of some rule in R and let $\sigma(q_{i1}(x_1) \ldots q_{ik}(x_k))$, $1 \leq i \leq m$, be all its right-hand sides. Introduce m new universal states $p_1, \ldots, p_m$ and the following transitions:

$$\delta(q, \sigma) = \{(p_1, 0), \ldots, (p_m, 0)\}, \qquad \delta(p_i, \sigma) = \{(q_{i1}, 1), \ldots, (q_{ik}, k)\}, \quad 1 \leq i \leq m,$$

such that q is universal in N iff it is universal in M. For $q \in Q$ and $\sigma \in \Sigma_0$, if $q(\sigma) \to \sigma$ is in R, then we let $\delta(q, \sigma) = \{(q_F, -1)\}$. Finally, we add $\delta(q_F, \sigma) = \{(q_F, -1)\}$ for every $\sigma \in \Sigma$. Repeating this procedure for every possible left-hand side of a rule of R, we obtain the set P of new states (all universal) as well as the transition function δ. It is easy to see that N faithfully simulates M and that $L(N) = L(M)$. The theorem now follows. □

Exactly the same proof applies if the original automaton M is utta; N is then a 2uta.

Corollary 4.5. UTTL $\subseteq$ 2UTL.

5. Alternating two-way synchronized pushdown tree automata

Two-way pushdown tree automata have the control structure of two-way finite tree automata while the operation of their storage, the pushdown, is synchronized with the movements of the automaton up and down the tree. We consider two different mechanisms for achieving synchronization. One, first studied in [7] in the context of tree transducers (see also [13]) will be the subject of this section; the other was introduced in [11] and it will be discussed in the next section. We study the effect of alternation and universality on these automata.

Definition 5.1. An *alternating two-way synchronized pushdown tree automaton* (2aspta) is a construct $M = (Q, U, \Sigma, \Gamma, \delta, q_0, z_0, F)$, where Q is a finite nonempty set of states, $U \subseteq Q$ is the set of *universal* states (states in $Q - U$ are called *existential* states), Σ is a ranked input alphabet, Γ is a pushdown alphabet, $q_0 \in Q$ is the initial state, $z_0 \in \Gamma$ is the bottom (initial) pushdown symbol, $F \subseteq Q$ is the set of accepting states, and δ is the transition function $\delta: Q \times \Sigma \times \Gamma \to \mathcal{P}(Q \times D)$, where $D = \{-1\} \cup \{(0, \gamma) \mid \gamma \in \Gamma\} \cup \{(i, \gamma_1\gamma_2) \mid 1 \leq i \leq m, \gamma_1, \gamma_2 \in \Gamma\}$ with m being the maximal j with $\Sigma_j \neq \emptyset$. Intuitively, D specifies the direction of move and the pushdown instruction: -1 means "move up on the tree and simultaneously pop the pushdown", $(0, \gamma)$ means "stay at the same node on the tree and simultaneously replace the top of the stack by γ" and $(i, \gamma_1\gamma_2)$ means "move down to the ith son on the tree and simultaneously replace the top cell of the pushdown by two cells $\gamma_1\gamma_2$"; in the last case, γ_1 is the new top of the pushdown.

An *instantaneous description* (ID) of M on a tree t in T_Σ is a quadruple of the form (q, α, β, t) where $q \in Q$, α is a simple path: $\alpha = (n_1, \ldots, n_k)$ with n_1 the root of t, n_k the node currently scanned, and n_{i+1} the son of n_i in t $(1 \leq i < k)$, $\beta = \gamma_1 \ldots \gamma_k \in \Gamma^*$ is the contents of the pushdown store $(\gamma_i \in \Gamma)$. A *universal* (*existential*) ID is an ID (q, α, β, t) with $q \in U$ $(q \in Q - U)$; it is *accepting* (*rejecting*) if $\alpha = (\ )$, $\beta = \lambda$, and $q \in F$ $(q \in Q - F)$. Let $I = (q, (n_1, \ldots, n_k), \gamma\beta_1, t)$ and $J = (p, (n_1, \ldots, n_l), \beta_2, t)$ be two ID's with n_k labeled by σ and $\gamma \in \Gamma$. Then, $I \vdash_M J$ if either of the following holds:

(1) $(p, (j, \gamma_1\gamma_2)) \in \delta(q, \sigma, \gamma)$; $\beta_2 = \gamma_1\gamma_2\beta_1$, $l = k+1$, and n_l is the jth ($j \geq 1$) son of n_k.

(2) $(p, (0, \gamma_1)) \in \delta(q, \sigma, \gamma)$; $\beta_2 = \gamma_1\beta_1$, $l = k$, and $n_l = n_k$.

(3) $(p, -1) \in \delta(q, \sigma, \gamma)$; $\beta_2 = \beta_1$, $l = k-1$, and $n_l = n_{k-1}$ if $k > 1$; if $k = 1$, then $I = (q, (n_1), \gamma, t) \vdash_M J = (p, (\ ), \lambda, t)$.

The reflexive, transitive closure of $\vdash_M$ is denoted by $\vdash_M^*$. The concepts of computation tree, accepting (rejecting) computation tree (of M on t), acceptance and the tree language recognized by a 2as-pta are defined exactly as in the section on alternating two-way finite tree automata (Section 4), except that configurations now are different. Also, the four varieties of automata, their names and the classes of languages they characterize are defined in an analogous fashion. For example, 2US-PTL is the family of all tree languages recognizable by 2us-pta's, universal two-way synchronized pushdown tree automata.

From Theorem 4.4 it follows that NTTL = RECOG $\subseteq$ ATTL $\subseteq$ 2ATL. It is also known that NTTL = RECOG = 2NS-PTL [13, Theorem 6.2] and it is obvious that 2ATL $\subseteq$ 2AS-PTL. We will now show 2AS-PTL $\subset$ 2NS-PTL, implying that all these classes are equal (to RECOG).

Our proof will be based on a simulation of a 2as-pta by a 2ns-pta. We will need some notation. Let $M = (Q, U, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a 2as-pta. Recall that $Q \times D$ is the set of all instructions of M and let $D' = D - \{-1\}$; then $\mathcal{P}(Q \times D')$ is the set of all sets of instructions, excluding 'move-up' instructions. We define a new alphabet

$$\Delta = \{\langle \phi, s, z \rangle \mid \phi \in \mathcal{P}(Q \times D'), s \subseteq Q, z \in \Gamma\}$$

and a 2ns-pta

$$N = (\mathcal{P}(Q), \emptyset, \Sigma, \Delta, \hat{\delta}, \{q_0\}, \langle \emptyset, \emptyset, Z_0 \rangle, \mathcal{P}(F) - \{\emptyset\}),$$

where $\hat{\delta}$ will be defined by means of a program written in pidgin-ALGOL (together with some English expressions). Symbols in the pushdown store are of the form $\langle \phi, s, z \rangle$, where $\phi \in \mathcal{P}(Q \times D')$, $s \subseteq Q$, and $z \in \Gamma$; we will have four variables ϕ, S, Z, and σ ranging respectively over $\mathcal{P}(Q \times D')$, $\mathcal{P}(Q)$, Γ, and Σ, and they will always refer to the respective values of the three components of the topmost pushdown symbol and the label of the currently scanned node of the input tree.

Suppose $\delta(q, \sigma, A) = \{(p_1, (i_1, \alpha_1)), \ldots, (p_k, (i_k, \alpha_k)), \quad (q_1, -1), \ldots, (q_l, -1)\}$, where $i_j \geq 0$ $(1 \leq j \leq k)$. We will use the following two functions:

$$\text{DOWN}(\delta(q, \sigma, A)) = \{(p_1, (i_1, \alpha_1)), \ldots, (p_k, (i_k, \alpha_k))\},$$

$$\text{UP}(\delta(q, \sigma, A)) = \{q_1, \ldots, q_l\}.$$

The function CHOOSE(B) nondeterministically chooses an element of the set B. The pushdown store is initialized to $\langle \emptyset, \emptyset, Z_0 \rangle$.

N simulates M by storing in the pushdown all the moves that still have to be taken. At any node of the input tree, N first tries all the computations down the tree (using the first component of the current pushdown symbol) and only when those are exhausted, N moves up in state that is the second component of the

pushdown symbol. Procedure UPDATE updates these two components taking into account the universal or existential nature of states.

Procedure UPDATE(q)

```
        if q ∈ U then φ ← φ ∪ DOWN(δ(q, σ, Z));
                      S ← S ∪ UP(δ(q, σ, Z))
                 else // q existential //
                      m ← CHOOSE(δ(q, σ, Z));
                      if m ∈ UP(δ(q, σ, z)) and m = (p, −1) then S ← S ∪ {p}
                          else φ ← φ ∪ {m}
                      fi
        fi
end UPDATE

// main program //

UPDATE(q0);

while STACK nonempty do
    case φ of
        = ∅ → pop pushdown and simultaneously move up in state S;
              for q ∈ S do UPDATE(q) od;
        ≠ ∅ → m = (p, (i, α)) ← CHOOSE(φ);
              φ ← φ − {m};
              if i = 0 then Z ← α; UPDATE(p)
                  else let α = γ1γ2; Z ← γ2;
                       move down to the ith son in state {p}
                       and simultaneously push ⟨∅, ∅, γ1⟩;
                        UPDATE(p)
              fi
    esac
od
```

We leave it to the reader to convince oneself that this program correctly simulates M and can be realized by a transition function $\hat{\delta}$ of a 2ns-pta. We have now proved the following theorem.

Theorem 5.2. 2AS-PTL $\subset$ 2NS-PTL.

Corollary 5.3. 2ATL = NTTL = ATTL = 2NS-PTL = 2DS-PTL = 2US-PTL = 2AS-PTL = RECOG.

Proof. Only the equality 2DS-PTL = 2NS-PTL was not mentioned before. This has been proved in [13]. □

6. Alternating two-way backtracking pushdown tree automata

Deterministic two-way backtracking pushdown automata (2db-pta) were introduced in [11]. These automata are similar to 2as-pta's with the following important difference. The pushdown has two tracks; a standard one for storing information for future reference, and an extra track for storing pointers to the nodes of the input tree. The machine can push the pushdown store while moving up or down the tree; at that time, the (pointer to the) node is pushed along with a symbol of the pushdown alphabet. When popping, the automaton 'backtracks', that is, continues its operation at the node that appears at the top of the pushdown immediately after the pop.

In this section, 2db-pta's are extended by adding the features of alternation, nondeterminism, and universality. Since some of the proofs of the results in this section are similar to those given either in [11] or in a previous section of this paper, we will mainly outline our arguments.

Definition 6.1. An *alternating two-way backtracking pushdown tree automaton* (2ab-pta) is a construct $M = (Q, U, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where Q is a finite, nonempty set of states, $U \subseteq Q$ is the set of *universal* states (states in $Q - U$ are called *existential*), Σ is a ranked input alphabet, Γ is the pushdown alphabet, $q_0 \in Q$ is the initial state, $Z_0 \in \Gamma$ is the bottom (initial) pushdown symbol, $F \subseteq Q$ is the set of accepting states, and δ is the transition function $\delta: Q \times \Sigma \times \Gamma \to \mathcal{P}(Q \times D)$, where (the instruction set) $D = \{\text{pop}\} \cup \{(i, \alpha) \mid -1 \leq i \leq m$, and $|\alpha| = 1$ if $i = 0$ and $|\alpha| = 2$ otherwise$\}$ with m the maximal j such that $\Sigma_j \neq \emptyset$.

An *instantaneous description* (ID) of M on a tree t in T_Σ is a quadruple (q, α, β, t) where $q \in Q$, α is a (nonnecessarily simple) path in t: $\alpha = (n_1, \ldots, n_k)$ with n_1 the root of t and n_k the currently scanned node, and $\beta = \gamma_1 \ldots \gamma_k \in \Gamma^*$ is the contents of the pushdown store ($\gamma_i \in \Gamma$). Universal, existential, accepting, and rejecting ID's are defined in exactly the same way as for 2as-pta's in the previous section.

For two ID's $I = (q, (n_1, \ldots, n_k), \gamma\beta_1, t)$ and $J = (p, (n_1, \ldots, n_l), \beta_2, t)$ with n_k labeled σ and $\gamma \in \Gamma$ we define $I \vdash_M J$ if either of the following holds:

(1) $(p, (j, \gamma_1\gamma_2)) \in \delta(q, \sigma, \gamma)$; $\beta_2 = \gamma_1\gamma_2\beta_1$, $l = k+1$, and n_l is the jth son of n_k if $j \geq 1$ or the father of n_k if $j = -1$.

(2) $(p, (0, \gamma_1)) \in \delta(q, \sigma, \gamma)$; $\beta_2 = \gamma_1\beta_1$, $l = k$, and $n_l = n_k$.

(3) $(p, \text{pop}) \in \delta(q, \sigma, \gamma)$; $\beta_2 = \beta_1$, $l = k-1$, and $n_l = n_{k-1}$ if $k > 1$; if $k = 1$, then $I = (q, (n_1), \gamma, t) \vdash_M J = (p, (\ ), \lambda, t)$.

The reflexive, transitive closure of $\vdash_M$ is denoted by $\vdash_M^*$. The concepts of computation tree, accepting (rejecting) computation tree, and acceptance are defined as in Section 5 (and in Section 4). The nondeterministic, universal, deterministic variants of 2ab-pta's are denoted respectively by 2nb-pta, 2ub-pta, 2db-pta and the corresponding families of automata and languages are denoted using the same conventions as we have used so far. For example, 2UB-PTL is the class of all tree languages recognizable by 2ub-pta's.

In [11] it is shown that 2DB-PTL = RECOG. In this section we show that all four classes of two-way bracktracking pushdown tree automata are equipotent, i.e., they all recognize RECOG. Instead of having one very involved construction we break our argument into three parts. The proof of part three, Theorem 6.4, is an adaptation of a proof in [11] where it is shown that any one-state 2db-pta can be simulated by a 2uta.

Theorem 6.2. *For any* 2ab-pta *there is an equivalent* 2nb-pta.

Lemma 6.3. *For any* 2nb-pta *there is an equivalent one-state* 2nb-pta.

Theorem 6.4. *For any one-state* 2nb-pta *there is an equivalent* 2ata.

Proofsketch of Theorem 6.2. The proof is similar to the proof of Theorem 5.2 with the following differences:

(i) The structure of the pushdown store is slightly different. The simulating machine N will have its pushdown symbols consist of four components, the additional component will store the (pointers to the) nodes of the input tree. This has to be taken care of during simulation.

(ii) Instead of functions DOWN and UP we should have now similarly (but not identically) defined functions PUSH and POP (respectively). The former should include all the moves except the pop-moves; the latter should store the pop-moves.

(iii) The main program should be modified to incorporate the possibility of pushing and simultaneous moving up (in the input tree) and, for the pop situation, the automaton should be able to move up or down the tree depending on the (pointer to the) node in the pushdown store (after the pop). □

Proof of Lemma 6.3. Without loss of generality we may assume that the 2nb-pta has a single accepting state and has no stay-moves (i.e. in each transition, the automaton moves either up or down the tree). Let $M = (Q, \emptyset, \Sigma, \Gamma, \delta, q_1, Z_0, \{q_F\})$ be the given 2nb-pta. We construct $N = (\{s\}, \emptyset, \Sigma, \Delta, \hat{\delta}, s, Z_0, \{s\})$, where $\Delta = \{Z_0\} \cup Q \times \Gamma \times Q$ and $\hat{\delta}$ is defined as follows:

(a) If $(p, (i, \gamma_1\gamma_2)) \in \delta(q_0, \sigma, Z_0)$, then $(s, (i, \langle p, \gamma_1, q\rangle\langle q, \gamma_2, q_F\rangle)) \in \hat{\delta}(s, \sigma, Z_0)$ for all $q \in Q$.

(b) If $(p, \text{pop}) \in \delta(q_0, \sigma, Z_0)$, then $(s, \text{pop}) \in \hat{\delta}(s, \sigma, Z_0)$.

(c) If $(q', (i, \gamma_1\gamma_2)) \in \delta(q, \sigma, Z)$, then $(s, (i, \langle q', \gamma_1, p'\rangle\langle p', \gamma_2, p\rangle)) \in \hat{\delta}(s, \sigma, \langle q, Z, p\rangle)$ for all p and p' in Q.

(d) If $(p, \text{pop}) \in \delta(q, \sigma, Z)$, then $(s, \text{pop}) \in \hat{\delta}(s, \sigma, \langle q, Z, p\rangle)$.

Intuitively, N keeps the finite control information of M on its pushdown store, including a guess as to the state after the current pushdown square is popped. □

Outline of proof of Theorem 6.4. Let $M = (\{s\}, \emptyset, \Sigma, \Gamma, \delta, s, Z_0, \{s\})$ be the given one-state 2nb-pta. We can assume without loss of generality that M has no stay instructions. For each $\sigma \in \Sigma$ and $Z \in \Gamma$ we construct a finite visit-tree that represents all the possible subsequent visits to a node labeled σ (in the input tree) and to a pushdown square from the time it is pushed (and then it carries Z) until it is popped.

(i) The root of the visit-tree is labeled by Z.

(ii) Let a node of the visit-tree be labeled by Y (in Γ).

(a) If $(s, (i, \gamma_1\gamma_2)) \in \delta(s, \sigma, Y)$, then create a son of Y and label it by γ_2.

(b) If $(s, \text{pop}) \in \delta(s, \sigma, Y)$, then create a son of Y and label it "pop".

(c) For each son of Y that is not labeled by "pop", check if its label occurs on the path from the root of the visit-tree to the node labeled by Y. If it does, relabel that node by "loop".

Clearly, the visit-tree corresponding to σ and Z is a finite tree whose leaves are labeled by "pop", "loop", or $A \in \Gamma$ for which $\delta(s, \sigma, A) = \emptyset$.

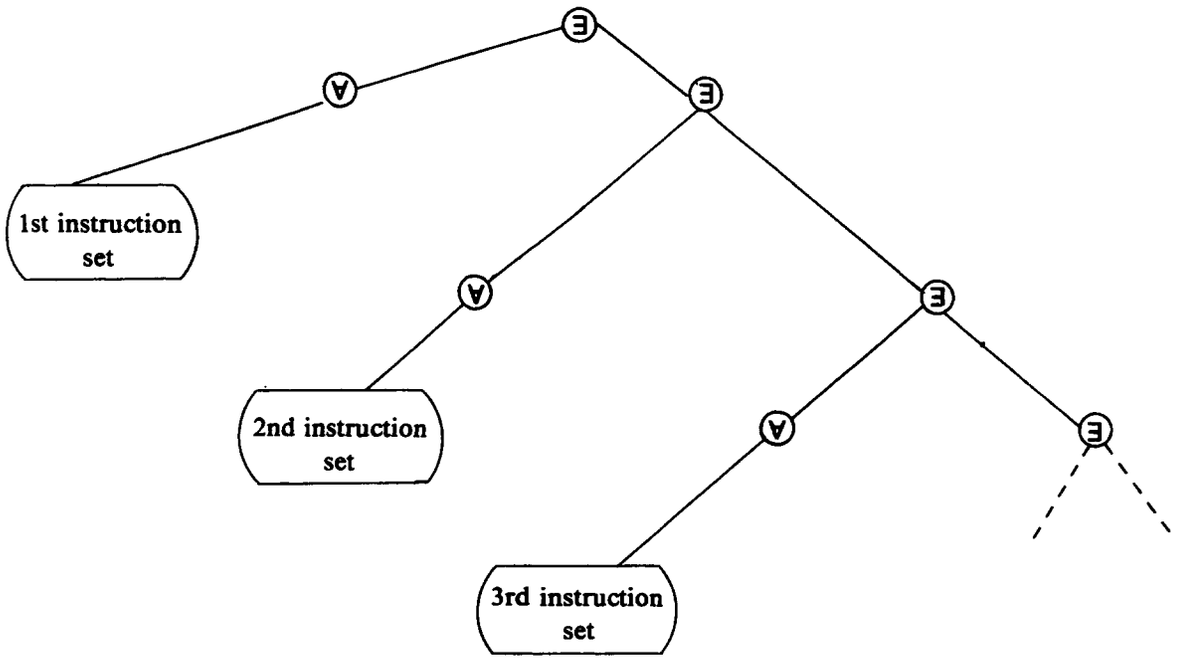


Fig. 4.

Let $\Pi_{\sigma,Z}$ be the set of all root-to-pop-leaf paths through the visit-tree corresponding to σ and Z. For $\pi \in \Pi_{\sigma,Z}$, $l(\pi)$ is the sequence of labels of the nodes along π; thus, $l(\pi)$ is of the form $Z\alpha(\text{pop})$, where $\alpha \in \Gamma^*$. Now, let $l(\pi) = ZZ_1 \ldots Z_k(\text{pop})$ and let the sequence of moves 'defining' π in the visit-tree be as follows:

$$(s, (i_1, Z_1'Z_1)) \in \delta(s, \sigma, Z),$$

$$(s, (i_2, Z_2'Z_2)) \in \delta(s, \sigma, Z_1),$$

$$\vdots$$

$$(s, (i_k, Z_k'Z_k)) \in \delta(s, \sigma, Z_{k-1}),$$

$$(s, \text{pop}) \in \delta(s, \sigma, Z_k).$$

Think of (i_j, Z_j') as an instruction for the 2ata saying: go to i_jth son if $i_j \geq 1$ and to the father if $i_j = -1$ in state Z_j'. Then, π induces a set of instructions

$$\text{instr}(\pi) = \{(i_1, Z_1'), \ldots, (i_k, Z_k')\}$$

that should lead to successful termination (i.e., they should be and-ed). Different paths of $\Pi_{\sigma,Z}$ induce different instruction sets which should be or-ed between themselves. All these instructions should be organized by introducing new existential and universal states (different for each $\sigma \in Z$ and $Z \in \Gamma$) according to Fig. 4, where $\exists$ ($\forall$) in a node means a new existential (universal) state. □

Corollary 6.5. 2DB-PTL = 2UB-PTL = 2NB-PTL = 2AB-PTL = RECOG.

7. Summary

In this paper we have studied the effect of alternation on several varieties of tree automata. We have seen that once the pushdown facility is available (either synchronized or backtracking), alternation provides no additional power. In the absence of pushdown, alternation (and universality) define new classes of tree languages. Altogether, we have distinguished between five families of tree languages as shown in Fig. 5. The diagram is correct because the finite language $\{S(ab), S(ba)\}$ is in 2DTL and not in DTTL, while the language L of Example 4.2 is not in 2NTL but certainly in DTTL.

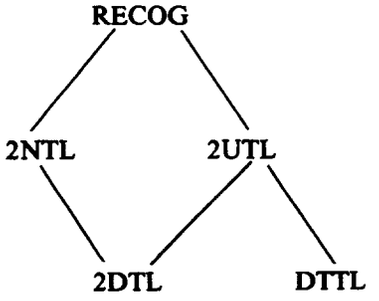


Fig. 5.

In [3] it is proved that alternating finite state automata define only regular sets. Our results have a similar flavor with respect to recognizable tree languages.

Acknowledgment

I would like to thank Tsutomu Kamimura for helpful discussions.

References

[1] L. Berman, Precise bounds for Presburger arithmetic and the reals with addition, *18th Symp. FOCS* (1977) 95–99.
[2] A.K. Chandra and L.J. Stockmeyer, Alternation, *17th Symp. FOCS* (1976) 98–108.
[3] A.K. Chandra, D.C. Kozen and L.J. Stockmeyer, Alternation, *J. ACM* **28** (1981) 114–133.
[4] J. Doner, Tree acceptors and some of their applications, *J. Comput. System Sci.* **4** (1970) 406–451.
[5] J. Engelfriet, *Tree Automata and Tree Grammars*, Lecture Notes DAIMI **FN-10** (University of Aarhus, Denmark, 1975).
[6] J. Engelfriet, Bottom-up and top-down tree transformations—A comparison, *Math. Systems Theory* **9** (1975) 193–231.
[7] J. Engelfriet, G. Rozenberg and G. Slutzki, Tree transducers, L-systems, and two-way machines, *J. Comput. System Sci.* **20** (1980) 150–202.
[8] M.J. Fischer and R.E. Ladner, Propositional modal logic of programs, *9th ACM STOC* (1977) 286–294.
[9] D.C. Kozen, On parallelism in Turing machines, *17th Symp. FOCS* (1976) 89–97.
[10] D.C. Kozen, Complexity of boolean algebras, *Theoret. Comput. Sci.* **10** (1980) 221–247.
[11] T. Kamimura, Tree automata and attribute grammars, Tech. Rept. TR-82-1, Dept. of Computer Science, Univ. of Kansas, Lawrence, KS, 1982.
[12] T. Kasai, A. Adachi and S. Iwata, Classes of pebble games and complete problems, *SIAM J. Comput.* **8** (1979) 574–586.
[13] T. Kamimura and G. Slutzki, Parallel and two-way automata on directed ordered acyclic graphs, *Inform. and Control* **49** (1981) 10–51.
[14] R.E. Ladner, R.J. Lipton and L.J. Stockmeyer, Alternating pushdown automata, *19th Symp. FOCS* (1978) 92–106.
[15] M. Magidor and G. Moran, Finite automata over finite trees, Tech. Rept. TR-69-30, Dept. of Mathematics, Hebrew Univ., Jerusalem, 1969.
[16] M.O. Rabin, Mathematical theory of automata, *Proc. Symp. Applied Mathematics* **19** (American Mathematical Society, Providence, RI, 1968) 153–175.
[17] G. Slutzki, Alternating, nondeterministic and universal pushdown automata, Tech. Rept. TR-82-2, Dept. of Computer Science, Univ. of Kansas, Lawrence, KS, 1982.
[18] L.J. Stockmeyer and A.K. Chandra, Provably difficult combinatorial games, *SIAM J. Comput.* **8** (1979) 151–174.
[19] J.W. Thatcher, Tree automata: Informal survey, in: A.V. Aho, ed., *Currents in the Theory of Computing* (Prentice-Hall, Englewood Cliffs, NJ, 1973) 143–172.
[20] J.W. Thatcher and J.B. Wright, Generalized finite automata theory with an application to a decision problem of second order logic, *Math. Systems Theory* **2** (1968) 57–81.