数理解析研究所講究録
第796巻 1992年 22-30

On Tree Automata and Partitioning Automata

東京女子大学（文理） 守屋悦朗 (Etsuro MORIYA)

Abstract

木オートマトンが受理するlabeled treeの集合は文脈自由文法の導出木によって特徴付けられることはよく知られている[Thatcher,5]。木オートマトンはstring上で動作する有限オートマトンをlabeled treeの上に拡張したものであるから、プッシュダウンオートマトンのそのような拡張も当然考えられる。このshort noteにおいてはlabeled tree上で動作するプッシュダウン木オートマトンを導入し、プッシュダウン木オートマトンが受理するlabeled treeの集合はインデックス文法の導出木によって特徴づけられることを示す。一方、ある種の並列計算のモデルとして導入されたpartitioning automaton[3]はstring上で動作するオートマトンであるが、受理されるinput stringはそのcompuation tree上をdepth-firstにたどったときのラベルの列になる。このことを使って、インデックス言語のもうひとつの特徴付けを得ることができる。

1 Introduction

A tree automaton is a finite automaton operating on labeled rooted trees. It is well-known [5] that the sets of labeled trees recognizable by tree automata can be characterized in terms of derivation trees for context-free grammars. In this note a generalization of tree automaton, tree automaton with pushdown memory, is introduced. It is shown that each of the sets of labeled trees recognizable by the automata is isomorphic to the set of derivation trees for some indexed grammar such that there is a homomorphism which renames the labels of the tree, and vice verse. It follows that an indexed language is a homomorphic image of the set of frontiers of labeled trees recognizable by some pushdown tree automaton, and conversely.

Partitioning automata were first introduced in [3] as a modified version of alternating automata, and subsequently considered by the author [4] to characterize classes of context-free grammars with memory. The set of strings accepted by a partitioning automaton is essentially the set of strings of labels associated with the depth-first traverses on the computation trees of the automaton on input strings. Using the partitioning pushdown automaton, it is shown that the set of strings of labels associated with the depth-first traverses on the labeled trees recognizable by a pushdown tree automaton is an indexed language, and vice verse.

2 Tree automata

Let Σ be an alphabet. A Σ-*tree* is a rooted ordered tree whose nodes each is labeled with an element of $\Sigma \cup \{\lambda\}$. Note that a node can be labeled with λ, the empty string, which means that nothing is labeled at the node. Figure 1 shows an $\{a, b\}$-tree. For a tree t, $fr(t)$, the *frontier* of t, denotes the string of symbols labeling the leaves from left to right. For example, the frontier of the tree in Figure 1 is *bbaab*. For a set T of trees, $fr(T)$ is the set $\{fr(t) \mid t \in T\}$.

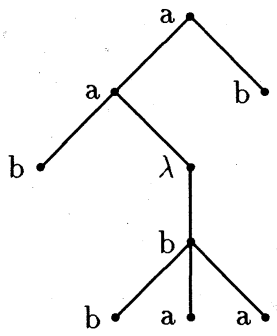

Figure 1: $\{a, b\}$-tree.

Now we define tree automaton with a worktape as a generalization of ordinary tree automaton. We use the 'root-to-frontier' model instead of the 'frontier-to-root' model [5]. A *tree automaton with a Turing worktape* is a sextuple $M = (K, \Sigma, \Gamma, \delta, s_0, F)$, where K is the finite set of *states*, Σ the finite set of *labels* for trees, Γ the finite set of *worktape symbols* including the *blank* B, $s_0 \in K$ the *initial state*, $F \subset K$ the set of *accepting states*, and δ the *transition function* mapping the elements of $K \times (\Sigma \cup \{\lambda\}) \times \Gamma$ into the finite subsets of $\bigcup_n (K \times \Gamma \times \{-1, 1\})^n$, where the superscript n means Cartesian product. An *ID* for M is an element of $K \times \Gamma^* \{\lceil\} \Gamma^*$, where $\lceil$, the read/write head on the worktape, is a special symbol not in Γ.

Given a labeled tree as an input, the automaton starts the computation to assign IDs to all nodes of the tree. To begin with, M assigns the *initial ID* $(s_0, \lceil B)$ to the root. Then generally, at a node which has exactly n children, is assigned ID $(p, \alpha \lceil Z \beta)$ and has label a, if $\delta(p, a, Z)$ contains $(q_1, Y_1, d_1; \ldots; q_n, Y_n, d_n)$, then M assigns IDs $(q_1, \gamma_1), \ldots, (q_n, \gamma_n)$ to the children of the node from left to right, where γ_i $(1 \leq i \leq n)$ is obtained from $\alpha \lceil Z \beta$ by replacing Z with Y_i and moving the worktape head $\lceil$ one symbol right or left according as d_i is 1 or -1. At a leaf, M computes IDs for the "hidden (unseen)" children of the leaf (the number of children of a leaf is arbitrary.) The input tree is *accepted* or *recognizable* by M if all IDs assigned to the hidden children are *accepting*, i.e., they all have accepting states. The set of trees accepted (recognizable) by M is denoted by $L(M)$.

The tree obtained from a tree accepted by M by relabeling each node with a pair of the Σ-symbol (which originally labels the node of the input tree) and an ID (which is assigned to the node according to the assignment procedure described above) is called a *computation tree*. The set of computation trees of M is denoted by $C(M)$.

We can consider tree automata with restricted types of worktape such as *tree automaton with a pushdown store* (*TPDA*), *tree automaton with counter(s)*, or *tree automaton with finite memory* (*TFA*). A TFA is nothing other than an ordinary root-to-frontier tree automaton. According to the conventional formulation for pushdown automata [2], however, in what

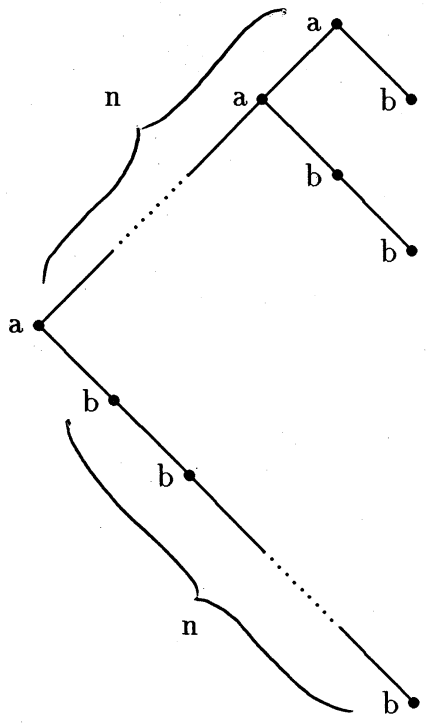

Figure 2: Tree not recognizable by any TFA.

follows we specify a TPDA as a 7-tuple $M = (K, \Sigma, \Gamma, \delta, s_0, Z_0, F)$, where Z_0 is a distinguished symbol of Γ to denote the 'bottom of stack', and δ is a mapping from $K \times (\Sigma \cup \{\lambda\})$ into the finite subsets of $\bigcup_n (K \times \Gamma^*)^n$. An ID for M is an element of $K \times \Gamma^*$ and the initial ID is (s_0, Z_0), where for an ID (p, γ), the leftmost symbol of γ serves as the topmost symbol of the stack. Unless stated otherwise, all notations used in this paper follow [2].

Example 1 Let M=($\{p, q, r, f\}$,$\{a\}$,$\{a, z\}$,δ, p, z,$\{f\}$) be a TPDA, where $\delta(p, a, z)$=$\{(q, az;$ $r, z)\}$, $\delta(q, a, a)$=$\{(q, aa; r, a), (r, a)\}$, $\delta(r, b, a)$=$\{(r, \lambda)\}$, and $\delta(r, b, z)$=$\{(f, \lambda)\}$. Then M accepts all and only those trees as shown in Figure 2. Figure 3 represents the unique computation tree for the tree in Figure 2, where the "hidden" children of leaves are shown with broken-line edges. $L(M)$ is not recognizable by any TFA, as we will see later.

Tree automata we defined so far are *nondeterministic* in the sense that the transition function permits more than one possible next moves at each step of computation. As usual we can define deterministic tree automata. For example, TPDA $M = (K, \Sigma, \Gamma, \delta, s_0, Z_0, F)$ is *deterministic* if

1) whenever $\delta(p, a, Z) \neq \emptyset$ for some a in Σ, then $\delta(p, \lambda, Z) = \emptyset$, and

2) for each p in K, a in $\Sigma \cup \{\lambda\}$ and Z in Γ, $\delta(p, a, Z)$ contains at most one element.

Let $\mathcal{L}_X$ denote the class of languages/sets of trees accepted by automata of type X which

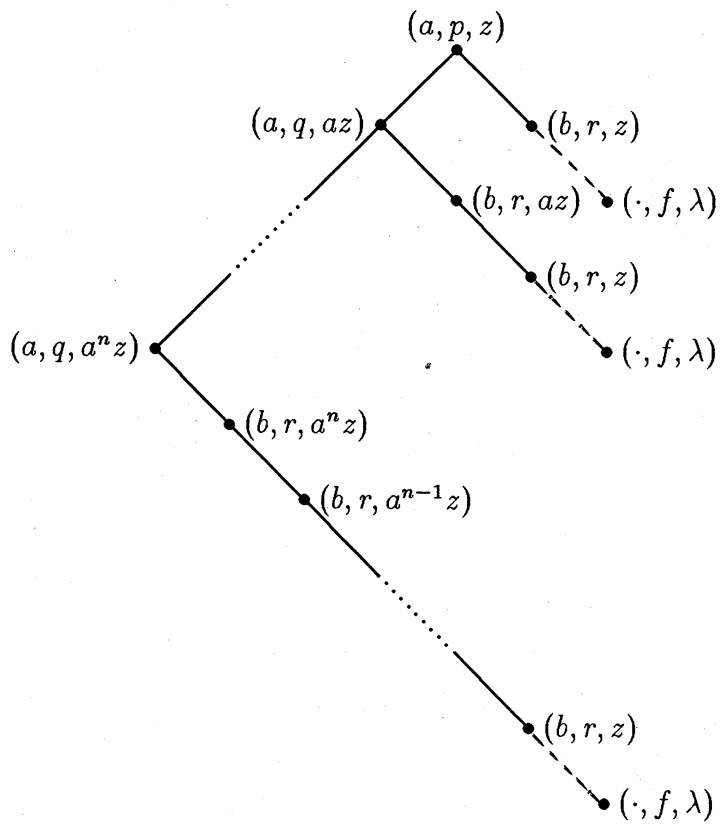

Figure 3: Computation tree.

operate on strings/trees. Consider unary labeled trees only. Then tree automata on those trees are essentially ordinary automata on strings. Thus we have

Theorem 2.1 *For classes X and Y of automata on strings, let X_{tree} and Y_{tree} denote classes of corresponding tree automata, respectively. If $\mathcal{L}_X \subsetneq \mathcal{L}_Y$ then $\mathcal{L}_{X_{tree}} \subsetneq \mathcal{L}_{Y_{tree}}$. In particular, $\mathcal{L}_{TFA} \subsetneq \mathcal{L}_{TPDA}$.*

A path from the root to a leaf in a labeled n-ary Σ-tree can be represented by a string over the alphabet $\Sigma^{(n)}=(\Sigma \cup \{\lambda\}) \times \{0, 1, \ldots, n\}$. For example, the path to the fourth leaf (from the left) of the tree in Figure 1 is represented by $(a, 1)(a, 2)(\lambda, 1)(b, 3)(a, 0)$. We call this string a *path string*. For a set of trees T, let $path(T)$ be the set of path strings for trees in T. Conversely, for a subset L of $(\Sigma^{(n)})^*$, let $path^{-1}(L)$ be the set of Σ-trees with the property that all path strings are in L. Note that $path^{-1}(path(T))$ is not necessarily T.

Magidor and Moran [5] showed that a set U of n-ary Σ-trees is recognizable by a deterministic TFA if and only if $U = path^{-1}(R)$ for a regular set R over $\Sigma^{(n)}$. To have a TPDA counterpart of this result, we need the following modification of the definition of path strings. Given an n-ary Σ-tree, consider a path string $(a_1, d_1) \cdots (a_l, d_l)(a_{l+1}, 0)$ as was defined above. Instead of this single string, we associate with the path the set of strings

$$(a_1, d_1) \cdots (a_l, d_l)(a_{l+1}, i),\ 1 \leq i \leq n.$$

Each of these strings is now called a path string. The following theorem holds under this definition of path string. A tree is *n-regular* if each of its interior nodes has exactly n children.

Theorem 2.2 *A set U of n-regular Σ-trees is recognizable by a deterministic TPDA if and only if $U = path^{-1}(L)$ for a deterministic context-free language L over $\Sigma^{(n)}$.*

A TPDA M is *obedient* if it has the following property: For any tree t, if all hidden children of a leaf, say π, can be assigned accepting IDs for M, then there is a supertree of t (i.e., a tree which is obtained from t by extending some leaf nodes, except for π, into trees) which is accepted by M.

Theorem 2.3 *For each TFA M, $path(L(M))$ is a regular set. For each obedient TPDA M, $path(L(M))$ is a context-free language.*

Consider the TPDA M in Example 1. Since $path(L(M)) = \{(a,1)^{n-1}(a,2)(b,1)^{n-1}(b,0) \mid n \geq 1\}$ is not a regular set, $L(M)$ is not recognizable by any TFA.

3 Indexed grammars

For our convenience, we slightly modify the definition of indexed grammar and its derivation trees. An *indexed grammar* is a quintuple $G = (N, \Gamma, \Sigma, P, S, z_0)$, where N is the finite set of *nonterminlas*, Γ the finite set of *stack symbols* such that $(N \cup \Sigma) \cap \Gamma = \emptyset$, Σ the finite set of *terminals* such that $N \cap \Sigma = \emptyset$, $S \subset N$ the set of *start symbols*, z_0 a distinguished symbol of Γ to denote the 'bottom of stack', and P the finite set of *productions* of the form

$$(*) \quad Af \rightarrow B_1\gamma_1 B_2\gamma_2 \cdots B_n\gamma_n,$$

where $n \geq 0$, A is in N, f in Γ, B_i $(1 \leq i \leq n)$ in $N \cup \Sigma$, γ_i in Γ^* if B_i is in N, and $\gamma_i = \lambda$ if B_i is in Σ.

Derivation trees for G, which are rooted trees whose nodes are labeled with elements of $(N \cup \Sigma \cup \{\lambda\}) \times \Gamma^*$, are defined recursively as follows.

i) The tree with single node whose label is (s, z_0) is a derivation tree, where s is an element of S.

ii) If t is a derivation tree such that t has a leaf π whose label is $(A, f\gamma)$, and if a production of the form (*), where $n \geq 1$, is in P, then the tree t' obtained from t by adding nodes $\pi_1, \pi_2, \ldots, \pi_n$ in this order from left to right as the children of π, is a derivation tree, where π_i $(1 \leq i \leq n)$ is labeled with $(B_i, \gamma_i\gamma)$.

iii) If t is a derivation tree, one of whose leaves, say π, has label $(A, f\gamma)$, and if $Af \rightarrow \lambda$ is a production in P, then the tree obtained from t by adding a node with label (λ, γ), as the unique child of π, is a derivation tree.

An *acceptable* derivation tree is a derivation tree whose leaves each is labeled with an element of $(\Sigma \cup \{\lambda\}) \times \Gamma^*$. The set of acceptable derivation trees for G is denoted by $T(G)$. Let h be the homomorphism mapping (a, γ) in $(\Sigma \cup \{\lambda\}) \times \Gamma^*$ into a. Then $L(G) = h(fr(T(G)))$ is an *indexed language*. It is easily seen that the indexed languages thus defined are precisely the indexed languages originally defined by Aho [1].

A *renaming* is a mapping from a (possibly infinite) alphabet to another. Let φ be a renaming from Σ to Δ. If t is a Σ-tree, then $\varphi(t)$ is a Δ-tree obtained from t by replacing the label, say a, of each node of t by $\varphi(a)$. In applying to strings over Σ, φ is considered to be a homomorphism from Σ^* to Δ^*.

Theorem 3.1 *For each indexed grammar G, there exists a TPDA M and a renaming φ such that $T(G) = \varphi(C(M))$.*

Theorem 3.2 *For each TPDA M, there exists an indexed grammar G and a renaming φ such that $C(M) = \varphi(T'(G))$, where $T'(G)$ is the set of trees obtained from the trees in $T(G)$ by removing all the leaves.*

Corollary 3.1 *For each TPDA M, $fr(L(M))$ is an indexed language.*

Corollary 3.2 *L is an indexed language if and only if $L = h(fr(C(M)))$ for some TPDA M and a homomorphism h.*

4 Partitioning automata

A partitioning automaton is very similar to an alternating automaton with one-way input tape. We begin with the definition of partitioning automaton having the most general type of memory. A *partitioning Turing machine* is specified by a 7-tuple $M = (K, U, \Sigma, \Gamma, \delta, s_0, F)$, where K is the finite set of *states*, $U \subseteq K$, Σ the finite set of *input symbols*, Γ the finite set of *worktape symbols*, $s_0 \in K$ the *initial state*, $F \subseteq K$ the set of *accepting states*, and δ the *transition function*, a mapping from $K \times \Sigma \times \Gamma$ into the subsets of $K \times \Gamma \times \{-1, 1\}$. An element of U is called a *partitioning state*, while an element of $K - U$ an *existential state*.

A *configuration* of M is an element of $K \times \Sigma^* \times \Gamma^* \{\lceil\} \Gamma^*$, where $\lceil$ is a symbol not in Γ. A configuration is *existential* (resp. *partitioning*) if its associated state is existential (resp. partitioning). An input word $w \in \Sigma^*$ is *accepted* by M if there exists a finite rooted tree t, called an *acceptable computation tree* on w, each of whose nodes is labeled with a configuration of M with the following properties:

(i) The root of t is labeled with the *initial configuration* $(s_0, w, \lceil B)$, where B, a symbol of Γ, is the *blank*.

(ii) Each leaf of t is labeled with an *accepting configuration*, a configuration of the form $(f, \lambda, \alpha\lceil\beta)$, where $f \in F$, and $\alpha, \beta \in \Gamma^*$.

(iii) **existential move** If π is an internal node whose label is $(p, ax, Z_1 \cdots \lceil Z_i \cdots Z_k)$, where p is in $K - U$, a in Σ and each Z in Γ, then π is called an *existential node* and has exactly one child whose label is either

(iii-1) $(q, x, Z_1 \cdots Y \lceil Z_{i+1} \cdots Z_k)$ if $\delta(p, a, Z_i)$ contains $(q, Y, 1)$, or
(iii-2) $(q, x, Z_1 \cdots \lceil Z_{i-1} Y \cdots Z_k)$ if $\delta(p, a, Z_i)$ contains $(q, Y, -1)$.

(iv) **partitioning move** If π is an internal node whose label is $(p, ax, Z_1 \cdots \lceil Z_i \cdots Z_k)$ with $p \in U$, and if $\delta(p, a, Z_i) = \{(q_1, Y_1, d_1), \cdots, (q_m, Y_m, d_m)\}$, then π is called a *partitioning node* and has exactly m children whose labels are (q_1, x_1, α_1), ..., (q_m, x_m, α_m) from left to right, where $x = x_1 \cdots x_m$ and each α_j is defined as in (iii).

In each of (iii) and (iv) above, a is consumed in the transition from node π to its children. Let t' and t'' be the trees obtained from t by replacing the label (i.e., an ID for M), say (p, ax, γ), of each node with label (p, a, γ) and a, respectively, where a is an element of $\Sigma \cup \{\lambda\}$ which is used in the transition from the node to its children. t' is called a *simplified* computation tree and t'' is called an *input tree* of t. The sets of acceptable simplified computation trees, input trees of acceptable computation trees, and input strings accepted by M are denoted by $C(M)$, $I(M)$, and $L(M)$, respectively.

The notion of partitioning automata was first introduced in [3] for pushdown automata and finite automata. A *partitioning pushdown automaton* (*PPDA*) is a partitioning Turing machine in which the worktape serves as a pushdown store, and a *partitioning finite automaton* (*PFA*) is a partitioning Turing machine in which no worktape symbols other than the blank can be written on the worktape. However, as we did for TPDA, we specify a PPDA by a 8-tuple $M = (K, U, \Sigma, \Gamma, \delta, s_0, Z_0, F)$, where Z_0, the bottom-of-stack symbol, is a distinguished symbol in Γ, and δ is a function from $K \times (\Sigma \cup \{\lambda\}) \times \Gamma$ into the finite subsets of K^*. A configuration for M is an element of $K \times \Sigma^* \times \Gamma^*$, and the initial configuration is (s_0, w, Z_0). A PFA is a PPDA such that if (q, γ) is in $\delta(p, a, Z)$ then $\gamma = Z$. Similarly we can define a partitioning counter automaton, a partitionig stack automaton etc., see [3][4].

Theorem 4.1 [3] *The class of languages accepted by PFAs is equal to the class of context-free languages and the class of languages accepted by PPDs is equal to the class of indexed languages.*

More relationships between classes of partitioning automata and classes of context-free grammars with memory are established in [4].

The tree in Figure 1 can be identified with the parenthesized expression

$$a(a(b, \Lambda(b(b, a, a)))b),$$

where Λ is a special symbol to denote λ. Furthermore, note that if the left parentheses are omitted from the expression and each of the right parentheses and the commas is replaced by a special symbol $\sharp$, then the resulting expression, $aab\sharp\Lambda bb\sharp a\sharp a\sharp\sharp\sharp\sharp b\sharp$, corresponds to the depth-first traverse of the tree, where $\sharp$ is used to indicate that a leaf is encountered in the traverse so that a traceback of lenght of the number of successive $\sharp$'s is to be caused. We call this string the *traverse representation* of the tree. Note that this correspondence between Σ-trees and strings over the alphabet $\Sigma \cup \{\sharp, \Lambda\}$ is one-to-one. The traverse representation of a tree t is denoted by $tr^{\sharp}(t)$. The string over Σ which is obtained from the traverse representation of a tree, say t, by deleting all occurrences of Λ or $\sharp$ is called the *traverse*

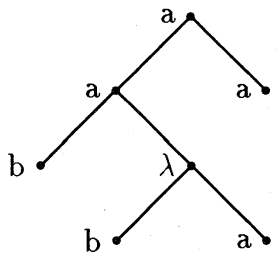

Figure 4: 2-regular $\{a, b\}$-tree.

string of the tree and denoted by $tr(t)$. For a set T of trees, let $tr^{\sharp}(T) = \{tr^{\sharp}(t) \mid t \in T\}$ and $tr(T) = \{tr(t) \mid t \in T\}$. An important observation is the following.

Lemma 4.1 *If M is a partitioning automaton, then $L(M) = tr(I(M))$.*

It can be shown that if M is a TFA (TPDA), then $tr^{\sharp}(L(M))$ is accepted by a partitioning automaton with a counter (partitioning automaton with a pushdown store and a counter, respectively). We do not know whether $tr^{\sharp}(L(M))$ is a context-free/indexed language or not when M is a TFA/TPDA. As for $tr(L(M))$, we have the following characterizations of the context-free languages and the indexed languages.

Theorem 4.2 *L is a context-free language if and only if $L = tr(L(M))$ for a TFA M. L is an indexed language if and only if $L = tr(L(M))$ for a TPDA M.*

In what follows we restrict ourselves to only n-regular trees. Given an n-regular Σ-tree t, a string over $\Sigma \cup \{\sharp, \Lambda\}$ is the *n-regular traverse string* of t, denoted by $tr^{(n)}(t)$, if it is obtained from the parethesized expression of t followed by a single comma, by replacing each comma in it with $\sharp$ and by deleting all parentheses. For example, $aab\sharp\Lambda b\sharp a\sharp a\sharp$ is the 2-regular traverse string of the tree in Figure 4. Note that $tr^{(n)}(t) = tr(t')$, where t' is the tree obtained from t by adding to each leaf a single child whose label is $\sharp$. It is important to note that the correspondence between n-regular Σ-trees and their n-regular traverse strings is one to one. Namely any n-regular Σ-tree can be identified with its n-regular traverse string. An *n-regular TPDA(TFA)* is a TPDA (TFA) which accepts only n-regular trees.

Theorem 4.3 *If M is an n-regular TFA, then $tr^{(2)}(L(M))$ is accepted by a PFA. If M is an n-regular TPDA, then $tr^{(2)}(L(M))$ is accepted by a PPDA.*

More generally it can be shown that if M is a tree automaton of type X, then $tr^{(n)}(L(M))$ is accepted by a partitioning automaton of type X. This theorem states that, under the coding by $tr^{(n)}$, any set of n-regular trees recognizable by a tree automaton of type X can be

defined by a partitioning automaton of type X. Thus partitioning automata can be viewed as a generalization of tree automata.

Acknowledgement

The author is grateful to Professor C.L.Liu for providing me an opportunity of spending a year at Department of Computer Science, University of Illinois at Urbana-Champaign, which made it possible for this paper to appear.

References

[1] Aho, A. V., Indexed grammars— an extension of context-free grammars, *J. Assoc. Comput. Mach.* **15** (1968) 647-671.

[2] Hopcroft, J. E. and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, (Addison-Wesley, Reading, Mass., 1979).

[3] Ikegawa, M. and T. Kasai, Modified one-way alternating pushdown automata and indexed languages, *Trans. IECE Japan* **E-69** (1986) 1213-1216.

[4] Moriya, E., Context-free grammars with memory, manuscript, 1990.

[5] Thatcher, J. W., Tree automata: an informal survey, in: A. V. Aho (ed.), *Currents in the Theory of Computing* (Prentice-Hall, Englewood Cliffs, N. J., 1973).