IEICE TRANS. INF. & SYST., VOL.E92–D, NO.2 FEBRUARY 2009

PAPER Special Section on Foundations of Computer Science

Linear-Time Recognizable Classes of Tree Languages by Deterministic Linear Pushdown Tree Automata

Akio FUJIYOSHI[†a)], ***Member***

SUMMARY In this paper, we study deterministic linear pushdown tree automata (deterministic L-PDTAs) and some variations. Since recognition of an input tree by a deterministic L-PDTA can be done in linear time, deterministic L-PDTAs are applicable to many kinds of applications. A strict hierarchy will be shown among the classes of tree languages defined by a variety of deterministic L-PDTAs. It will be also shown that deterministic L-PDTAs are weakly equivalent to nondeterministic L-PDTAs.

key words: *linear pushdown tree automaton, context-free tree grammar, tree structure, linear-time recognizability*

1. Introduction

Recently, the demand for processing extremely large amount of data has become greater, and thus linear-time recognizability is requested in many cases. For the process of validation of tree-structural data represented by XML documents [1], XML schema languages (DTD, W3C XML Schema, RELAX NG, etc.) [12] are commonly used. Since XML schema languages define tree languages that are linear-time recognizable, a number of efficient algorithms have been developed based on them. Though XML schema languages are very useful for many kinds of applications, they cannot handle context-free sequences. The class of tree languages defined by XML schema languages is included in the class of recognizable tree languages [12], that is, the class of tree language recognized by deterministic bottom-up tree automata [2]. If we want to process tree languages with context-free sequences, we have to think of more powerful formalisms.

In this paper, we study deterministic linear pushdown tree automata (deterministic L-PDTAs) and some variations. Since recognition of an input tree by a deterministic L-PDTA can be done in linear time, deterministic L-PDTAs are also useful for many kinds of applications. The following three types of deterministic L-PDTAs will be studied: (1) real-time deterministic L-PDTAs, (2) deterministic L-PDTAs, and (3) deterministic L-PDTAs with regular look-ahead. As desired, it will be shown that recognition by a deterministic L-PDTA of all three types can be done in linear time.

It is necessary to think of adding the function of 'regular look-ahead' to deterministic L-PDTAs. It is known that deterministic top-down tree automata are strictly less powerful than deterministic bottom-up tree automata [2]. Because an L-PDTA is a generalization of top-down tree automaton, there exist some basic tree languages that cannot be recognized by any deterministic L-PDTA. For example, the tree language $\{f(a, b), f(b, a)\}$ is neither recognized by a deterministic top-down tree automaton nor by a deterministic L-PDTA. With the function of 'regular look-ahead', deterministic L-PDTAs become strictly more powerful than deterministic bottom-up tree automata. The idea of 'regular look-ahead' was brought from the study of tree transducers [3].

To examine the recognition capability of deterministic L-PDTAs, we compare the classes of the three types of deterministic L-PDTAs and the class of nondeterministic L-PDTAs with regard to tree languages and yield languages. Concerning tree languages, it will be shown that a strict hierarchy holds among the classes of recognized tree languages. Concerning yield languages, on the other hand, it will be shown that the classes of their yield languages are the same, i.e., the classes of deterministic and nondeterministic L-PDTAs are all weakly equivalent. In the proof of these statements, the formal properties of linear monadic context-free tree grammars (LM-CFTGs) were utilized. LM-CFTGs are the grammar formalism corresponding to L-PDTAs.

This paper is organized as follows. In Sect. 2, terms, definitions, LM-CFTG and L-PDTA are introduced. In Sect. 3, three types of deterministic L-PDTAs are defined. In Sect. 4, the recognition capability of deterministic L-PDTAs is examined.

2. Preliminaries

In this section, terms, definitions, and former results which will be used in the rest of this paper are introduced.

Let $\mathcal{N}$ be the set of all natural numbers, and let $\mathcal{N}_+$ be the set of all positive integers. The concatenation operator is denoted by '$\cdot$'. For an alphabet Σ, the set of strings over Σ is denoted by Σ^*, and the empty string is denoted by λ.

2.1 Ranked Alphabets and Trees

A *ranked alphabet* is a finite set of symbols in which each symbol is associated with a natural number, called the *arity* of a symbol. Let Σ be a ranked alphabet. For $a \in \Sigma$, the arity of a is denoted by $\mathrm{arity}(a)$. For $n \geq 0$, let $\Sigma_n = \{a \in \Sigma \mid \mathrm{arity}(a) = n\}$.

Manuscript received March 28, 2008.
Manuscript revised June 20, 2008.

[†]The author is with the Department of Computer and Information Sciences, Ibaraki University, Hitachi-shi, 316–8511 Japan.
a) E-mail: fujiyosi@mx.ibaraki.ac.jp

DOI: 10.1587/transinf.E92.D.248

A set D is a *tree domain* if D is a nonempty finite subset of $(\mathcal{N}_+)^*$ satisfying the following conditions:

- For any $d \in D$, if $d', d'' \in (\mathcal{N}_+)^*$ and $d = d' \cdot d''$, then $d' \in D$.
- For any $d \in D$ and $i, j \in \mathcal{N}_+$, if $i \leq j$ and $d \cdot j \in D$, then $d \cdot i \in D$.

Let D be a tree domain, and let $d \in D$. Elements in D are called *nodes*. A node d' is a *child* of d if there exists $i \in \mathcal{N}_+$ such that $d' = d \cdot i$. A node is called a *leaf* if it has no child. The node λ is called the *root*. A node that is neither a leaf nor the root is called an *internal node*.

Let Σ be a ranked alphabet. A *tree* over Σ is a function $\alpha : D \to \Sigma$ where D is a tree domain and, for all $d \in D$, $\mathrm{arity}(\alpha(d)) = \max\{i \in \mathcal{N}_+ \mid d \cdot i \in D\}$. The set of trees over Σ is denoted by T_Σ. A subset of T_Σ is called a *tree language*. The domain of a tree α is denoted by D_α. For $d \in D_\alpha$, $\alpha(d)$ is called the *label* of d. The *subtree* of α at d is $\alpha/d = \{(d', a) \in (\mathcal{N}_+)^* \times \Sigma \mid (d \cdot d', a) \in \alpha\}$.

The expression of a tree over Σ is defined to be a string over elements of Σ, parentheses and commas. For $\alpha \in T_\Sigma$, if $\alpha(\lambda) = b \in \Sigma_n$, and, for each $1 \leq i \leq n$, the expression of α/i is α_i, then the expression of α is $b(\alpha_1, \alpha_2, \ldots, \alpha_n)$. Note that n is the number of the children of the root. For $b \in \Sigma_0$, trees are written as b instead of $b()$. When the expression of α is $b(\alpha_1, \alpha_2, \ldots, \alpha_n)$, it is written that $\alpha = b(\alpha_1, \alpha_2, \ldots, \alpha_n)$, i.e., each tree is identified with its expression.

Let Σ be a ranked alphabet, and let I be a set that is disjoint from Σ. $T_\Sigma(I)$ is defined to be $T_{\Sigma \cup I}$ where $\Sigma \cup I$ is the ranked alphabet obtained from Σ by adding all elements in I as symbols of arity 0.

Let $X = \{x_1, x_2, \ldots\}$ be the fixed countable set of variables. Let $X_0 = \emptyset$ and for $n \geq 1$, let $X_n = \{x_1, x_2, \ldots, x_n\}$. x_1 is situationally denoted by x.

Let $\alpha, \beta \in T_\Sigma$, and let $d \in D_\alpha$. We define $\alpha\langle d \leftarrow \beta\rangle = \{(d', a) \mid (d', a) \in \alpha \text{ and } d \text{ is not a prefix of } d'\} \cup \{(d \cdot d'', b) \mid (d'', b) \in \beta\}$, i.e., the tree $\alpha\langle d \leftarrow \beta\rangle$ is the result of replacing α/d by β.

Let $\alpha \in T_\Sigma(X_n)$, and let $\beta_1, \beta_2, \ldots, \beta_n \in T_\Sigma(X)$. The notion of substitution is defined. The result of substituting each β_i for nodes labeled by variable x_i in α, denoted by $\alpha[\beta_1, \beta_2, \ldots, \beta_n]$, is defined as follows:

- If $\alpha \in \Sigma_0$, then $\alpha[\beta_1, \beta_2, \ldots, \beta_n] = \alpha$.
- If $\alpha \in X_n$ and $\alpha = x_i$, then $\alpha[\beta_1, \beta_2, \ldots, \beta_n] = \beta_i$.
- If $\alpha = b(\alpha_1, \alpha_2, \ldots, \alpha_k)$ and $k \geq 1$, then
  $\alpha[\beta_1, \beta_2, \ldots, \beta_n]$
  $= b(\alpha_1[\beta_1, \beta_2, \ldots, \beta_n], \ \ldots\ , \alpha_k[\beta_1, \beta_2, \ldots, \beta_n])$.

Let ε be the special symbol that can be contained in Σ_0. The *yield* of a tree is a function from T_Σ into Σ_0^* defined as follows:

(1) If $\alpha \in (\Sigma_0 - \{\varepsilon\})$, then $\mathrm{yield}(\alpha) = \alpha$, or if $\alpha = \varepsilon$, then $\mathrm{yield}(\alpha) = \lambda$.
(2) If $\alpha = b(\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $n \geq 1$, then $\mathrm{yield}(\alpha) = \mathrm{yield}(\alpha_1) \cdot \mathrm{yield}(\alpha_2) \cdot \cdots \cdot \mathrm{yield}(\alpha_n)$.

For $L \subseteq T_\Sigma$, the *yield language of* L is the set $\mathrm{yield}(L) = \{\mathrm{yield}(\alpha) \mid \alpha \in L\}$.

Let Σ and Σ' be ranked alphabets such that $\Sigma_0 = \Sigma'_0$. Let $L \subseteq T_\Sigma$ and $L' \subseteq T_{\Sigma'}$. L and L' are *weakly equivalent* if $\mathrm{yield}(L) = \mathrm{yield}(L')$.

2.2 Context-Free Tree Grammars

The context-free tree grammars (CFTGs) were introduced by W.C. Rounds [13] as tree generating systems. The definition of CFTGs is a direct generalization of context-free grammars (CFGs) [10].

Definition 1: A *context-free tree grammar* (CFTG) is a four-tuple $\mathcal{G} = (N, \Sigma, P, S)$, where: N and Σ are disjoint ranked alphabets of *nonterminals* and *terminals*, respectively. P is a finite set of *rules* of the form $A(x_1, x_2, \ldots, x_n) \to \alpha$ with $n \geq 0$, $A \in N_n$, and $\alpha \in T_{N \cup \Sigma}(X_n)$. For $A \in N_0$, rules are written as $A \to \alpha$ instead of $A() \to \alpha$. S, the *initial nonterminal*, is a distinguished symbol in N_0.

For a CFTG $\mathcal{G}$, the *one-step derivation* $\underset{\mathcal{G}}{\Rightarrow}$ is the relation on $T_{N\cup\Sigma} \times T_{N\cup\Sigma}$ such that, for a tree $\alpha \in T_{N\cup\Sigma}$ and a node $d \in D_\alpha$, if $\alpha/d = A(\alpha_1, \alpha_2, \ldots, \alpha_n)$, $A \in N_n$, $\alpha_1, \alpha_2, \ldots, \alpha_n \in T_{N\cup\Sigma}$ and $A(x_1, x_2, \ldots, x_n) \to \beta$ is in P, then $\alpha \underset{\mathcal{G}}{\Rightarrow} \alpha\langle d \leftarrow \beta[\alpha_1, \alpha_2, \ldots, \alpha_n]\rangle$. See Fig. 1. The reflective transitive closure of $\underset{\mathcal{G}}{\Rightarrow}$ is denoted by $\overset{*}{\underset{\mathcal{G}}{\Rightarrow}}$. The *tree language generated by* $\mathcal{G}$ is the set $L(\mathcal{G}) = \{\alpha \in T_\Sigma \mid S \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \alpha\}$.

Linear monadic CFTGs (LM-CFTGs) are CFTGs with two restrictions. One, it is 'linear', which requires the number of occurrences of every variable in the right-hand side of a production be no more than 1; and two, it is 'monadic', which requires the arity of nonterminals be either 0 or 1. In [4], [5], it was shown that LM-CFTGs are weakly equivalent to tree adjoining grammars (TAGs) [11].

Definition 2: A CFTG $\mathcal{G} = (N, \Sigma, P, S)$ is *monadic* if the arity of any nonterminal is either 0 or 1, i.e., $N = N_0 \cup N_1$ and $N_n = \emptyset$ for $n \geq 2$. $\mathcal{G}$ is *linear* if for any production $A(x_1, x_2, \ldots, x_n) \to \alpha$ in P, no variable occurs more than once in α.

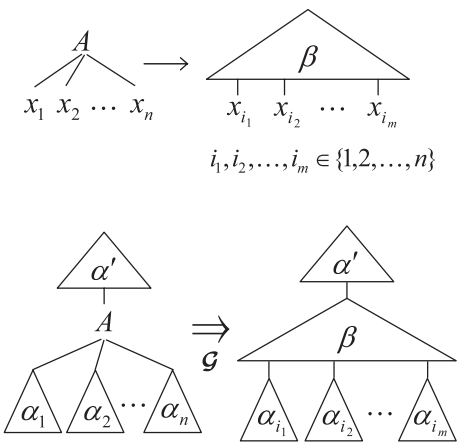

Fig. 1 One-step derivation.

2.3 Pushdown Tree Automata

Pushdown tree automata (PDTAs) [9] were introduced by I. Guessarian in order to formalize the class of tree languages generated by CFTGs. A PDTA can be seen as the combination of an ordinary pushdown finite automaton [10] and a top-down tree automaton [2].

Definition 3: A (nondeterministic) *pushdown tree automaton* (PDTA) is a six-tuple $M = (Q, \Sigma, \Gamma, q_0, Z_0, R)$, where Q is a finite set of *states*, Σ is a ranked alphabet, called the *input alphabet*, Γ is a ranked alphabet such that $\Gamma = \Gamma_0 \cup \Gamma_1$, called the *pushdown alphabet*, $q_0 \in Q$ is the *initial state*, $Z_0 \in \Gamma_0$ is the *start symbol*, and R is a finite set of *rules* of one of the following forms:

read rule:

(i) $q(a, A) \to a$ with $a \in \Sigma_0$, $q \in Q$ and $A \in \Gamma_0$.
(ii) $q(a, B) \to a$ with $a \in \Sigma_0$, $q \in Q$ and $B \in \Gamma_1$.
(iii) $q(b(x_1, x_2, \ldots, x_n), A) \to b(q_1(x_1, \pi_1), q_2(x_2, \pi_2), \ldots, q_n(x_n, \pi_n))$ with $n \geq 1$, $b \in \Sigma_n$, $q, q_1, q_2, \ldots, q_n \in Q$, $A \in \Gamma_0$ and $\pi_1, \pi_2, \ldots, \pi_n \in \Gamma_1^*\Gamma_0$.
(iv) $q(b(x_1, x_2, \ldots, x_n), B) \to b(q_1(x_1, \pi_1), q_2(x_2, \pi_2), \ldots, q_n(x_n, \pi_n))$ with $n \geq 1$, $b \in \Sigma_n$, $q, q_1, q_2, \ldots, q_n \in Q$, $B \in \Gamma_1$ and $\pi_1, \pi_2, \ldots, \pi_n \in \Gamma_1^*\Gamma_0 \cup \Gamma_1^*$.

ε-rule:

(v) $q(x, A) \to q'(x, \pi)$ with $q, q' \in Q$, $A \in \Gamma_0$ and $\pi \in \Gamma_1^*\Gamma_0$.
(vi) $q(x, B) \to q'(x, \pi)$ with $q, q' \in Q$, $B \in \Gamma_1$ and $\pi \in \Gamma_1^*$.

An *instantaneous description* of M is a triple $q(\alpha, \pi) \in Q \times T_\Sigma \times \Gamma_1^*\Gamma_0$. Let ID be the set of all instantaneous descriptions of M. A *configuration* of M is an element of $T_\Sigma(ID)$. The *move relation* $\vdash_M$ of M is the relation defined as follows. For any configurations $c, c' \in T_\Sigma(ID)$, $c \vdash_M c'$ if there exists a node $d \in D_c$ that satisfies one of the following conditions:

- A type (i) rule $q(a, A) \to a$ is in R, $c/d = q(a, A)$, and $c' = c\langle d \leftarrow a\rangle$.
- A type (ii) rule $q(a, B) \to a$ is in R, $c/d = q(a, B\rho)$ for some $\rho \in \Gamma_1^*\Gamma_0$, and $c' = c\langle d \leftarrow a\rangle$. (The content of stack ρ is discarded.)
- A type (iii) rule $q(b(x_1, \ldots, x_n), A) \to b(q_1(x_1, \pi_1), \ldots, q_n(x_n, \pi_n))$ is in R, $c/d = q(b(\alpha_1, \ldots, \alpha_n), A)$ for some $\alpha_1, \ldots, \alpha_n \in T_\Sigma$, and $c' = c\langle d \leftarrow b(q_1(\alpha_1, \pi_1), \ldots, q_n(\alpha_n, \pi_n))\rangle$.
- A type (iv) rule $q(b(x_1, \ldots, x_n), B) \to b(q_1(x_1, \pi_1), \ldots, q_n(x_n, \pi_n))$ is in R, $c/d = q(b(\alpha_1, \ldots, \alpha_n), B\rho)$ for some $\alpha_1, \ldots, \alpha_n \in T_\Sigma$ and $\rho \in \Gamma_1^*\Gamma_0$, and $c' = c\langle d \leftarrow b(q_1(\alpha_1, \pi_1'), \ldots, q_n(\alpha_n, \pi_n'))\rangle$ where for each $1 \leq i \leq n$, if $\pi_i \in \Gamma_1^*\Gamma_0$, then $\pi_i' = \pi_i$, and if $\pi_i \in \Gamma_1^*$, then $\pi_i' = \pi_i\rho$. (If $\pi_i, \pi_j \in \Gamma_1^*$ for some $i \neq j$, then the content of stack ρ is duplicated.)
- A type (v) rule $q(x, A) \to q'(x, \pi)$ is in R, $c/d = q(\alpha, A)$ and $\alpha \in T_\Sigma$, and $c' = c\langle d \leftarrow q'(\alpha, \pi)\rangle$.
- A type (vi) rule $q(x, B) \to q'(x, \pi)$ is in R, $c/d = q(\alpha, B\rho)$ for some $\alpha \in T_\Sigma$ and $\rho \in \Gamma_1^*\Gamma_0$, and $c' = c\langle d \leftarrow q'(\alpha, \pi\rho)\rangle$.

A *computation* is a finite sequence of configurations $c_1 c_2 \cdots c_n$ such that $n \geq 1$, $c_1, c_2, \ldots, c_n \in T_\Sigma(ID)$ and $c_1 \vdash_M c_2 \vdash_M \cdots \vdash_M c_n$. When there exists a computation $c_1 c_2 \cdots c_n$, we write $c_1 \vdash_M^* c_n$. The *tree language recognized by* M is the set $T(M) = \{\alpha \in T_\Sigma \mid q_0(\alpha, Z_0) \vdash_M^* \alpha\}$.

Let M and M' be PDTAs. M and M' are *equivalent* if $T(M) = T(M')$. M and M' are *weakly equivalent* if $\mathrm{yield}(T(M)) = \mathrm{yield}(T(M'))$. Let $\mathcal{M}$ and $\mathcal{M}'$ be classes of PDTAs. $\mathcal{M}$ and $\mathcal{M}'$ are *weakly equivalent* if $\{\mathrm{yield}(T(M)) \mid M \in \mathcal{M}\} = \{\mathrm{yield}(T(M')) \mid M' \in \mathcal{M}'\}$.

We introduce a restriction on PDTAs, called 'linear' [4], [6]. Linear PDTAs (L-PDTAs) don't have the capability of duplicating their pushdown stack. The class of tree languages recognized by L-PDTAs coincides with that generated by LM-CFTGs [4], [5].

Definition 4: A PDTA $M = (Q, \Sigma, \Gamma, q_0, Z_0, R)$ is *linear* if, for each type (iv) rule $q(b(x_1, x_2, \ldots, x_n), B) \to b(q_1(x_1, \pi_1), q_2(x_2, \pi_2), \ldots, q_n(x_n, \pi_n))$ in R, $|\{i \mid 1 \leq i \leq n \text{ and } \pi_i \in \Gamma_1^*\}| = 1$.

Example 1: The following M is an L-PDTA that accepts a tree language whose yield language is $L_{ww} = \{ww \mid w \in \{a, b\}^+\}$. $M = (Q, \Sigma, \Gamma, q_0, Z_0, R)$, where $Q = \{q_0, q_1, q_2, q_A, q_B\}$, $\Sigma = \Sigma_0 \cup \Sigma_2$, $\Sigma_0 = \{a, b\}$, $\Sigma_2 = \{d\}$, $\Gamma_0 = \{Z_0\}$, $\Gamma_1 = \{N, A, B\}$, and R consists of the following rules:

$q_0(x, Z_0) \to q_1(x, NZ_0)$,
$q_1(d(x_1, x_2), N) \to d(q_A(x_1, Z_0), q_1(x_2, NA))$,
$q_1(d(x_1, x_2), N) \to d(q_B(x_1, Z_0), q_1(x_2, NB))$,
$q_1(x, N) \to q_2(x, \lambda)$,
$q_2(d(x_1, x_2), A) \to d(q_2(x_1, \lambda), q_A(x_2, Z_0))$,
$q_A(a, Z_0) \to a$,
$q_2(d(x_1, x_2), B) \to d(q_2(x_1, \lambda), q_B(x_2, Z_0))$,
$q_B(b, Z_0) \to b$,
$q_2(x, A) \to q_A(x, \lambda)$, and
$q_2(x, B) \to q_B(x, \lambda)$.

In Fig. 2, a tree in $T(M)$ and the movement of M for the tree are illustrated.

3. Deterministic Linear Pushdown Tree Automata

In this section, we give the definitions of (1) real-time deterministic L-PDTAs, (2) deterministic L-PDTAs, and (3) deterministic L-PDTAs with regular look-ahead. These three classes of L-PDTAs are newly defined in this paper. The ideas of 'real-time' and 'deterministic' were brought from classic automata theory [10], and the idea of 'regular look-ahead' was brought from the study of tree transducers [3]. It will be shown that recognition by a deterministic L-PDTA

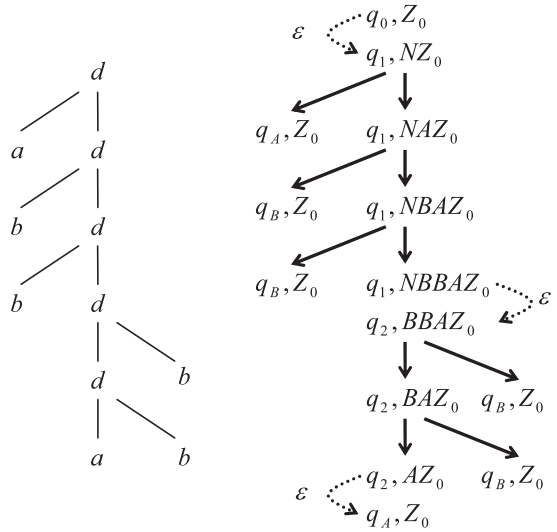

Fig. 2 An example of a tree accepted by M and its movement.

of all three types can be done in linear time.

The definitions of 'real-time' and 'deterministic' are as follows:

Definition 5: An L-PDTA $M = (Q, \Sigma, \Gamma, q_0, Z_0, R)$ is *real-time* if R doesn't contain any ε-rules.

Definition 6: An L-PDTA $M = (Q, \Sigma, \Gamma, q_0, Z_0, R)$ is *deterministic* if it satisfies the following conditions:

- For each $q \in Q$, $a \in \Sigma$ and $B \in \Gamma$, R contains at most one read rule with q, a and B on its left-hand side.
- For each $q \in Q$ and $B \in \Gamma$, R contains at most one ε-rule with q and B on its left-hand side.
- For each $q \in Q$ and $B \in \Gamma$, if R contains a ε-rule with q and B on its left-hand side, then for any $a \in \Sigma$, R doesn't contain any read rules with q, a and B on its left-hand side.

An L-PDTA with regular look-ahead is defined as follows:

Definition 7: An *L-PDTA with regular look-ahead* is a eight-tuple $M = (P, Q, \Sigma, \Gamma, q_0, Z_0, \delta, R)$, where P is a finite set of *look-ahead states*, Q, Σ, Γ, q_0 and Z_0 are as for an original L-PDTA, δ is a mapping from T_Σ into P realized by a deterministic bottom-up tree automaton, and R is a finite set of *rules* of one of the following forms:

read rule:

(i) $(p, q)(a, A) \to a$
with $a \in \Sigma_0$, $p \in P$, $q \in Q$ and $A \in \Gamma_0$.

(ii) $(p, q)(a, B) \to a$
with $a \in \Sigma_0$, $p \in P$, $q \in Q$ and $B \in \Gamma_1$.

(iii) $(p, q)(b(x_1, x_2, \ldots, x_n), A) \to b(q_1(x_1, \pi_1), q_2(x_2, \pi_2), \ldots, q_n(x_n, \pi_n))$
with $n \geq 1$, $b \in \Sigma_n$, $p \in P$, $q, q_1, q_2, \ldots, q_n \in Q$, $A \in \Gamma_0$ and $\pi_1, \pi_2, \ldots, \pi_n \in \Gamma_1^* \Gamma_0$.

(iv) $(p, q)(b(x_1, x_2, \ldots, x_n), B) \to b(q_1(x_1, \pi_1), q_2(x_2, \pi_2), \ldots, q_n(x_n, \pi_n))$
with $n \geq 1$, $b \in \Sigma_n$, $p \in P$, $q, q_1, q_2, \ldots, q_n \in Q$, $B \in \Gamma_1$ and $\pi_1, \pi_2, \ldots, \pi_n \in \Gamma_1^* \Gamma_0 \cup \Gamma_1^*$ such that $|\{i \mid 1 \leq i \leq n \text{ and } \pi_i \in \Gamma_1^*\}| = 1$.

ε-rule:

(v) $(p, q)(x, A) \to q'(x, \pi)$
with $p \in P$, $q, q' \in Q$, $A \in \Gamma_0$ and $\pi \in \Gamma_1^* \Gamma_0$.

(vi) $(p, q)(x, B) \to q'(x, \pi)$
with $p \in P$, $q, q' \in Q$, $B \in \Gamma_1$ and $\pi \in \Gamma_1^*$.

A recognition process of an L-PDTA with regular look-ahead M is done in two stages: look-ahead stage and recognition stage. In look-ahead stage, M assigns a look-ahead state to each node of an input tree as a deterministic bottom-up tree automaton does. For an input tree α and its node d, the look-ahead state $\delta(\alpha/d)$ will be assigned. And then, in recognition stage, M moves like an original L-PDTA except that M decides next move according to a look-ahead state in addition to a state and a pushdown symbol on the top.

The move relation is defined as for an original L-PDTA except that to process an instantaneous description $q(\alpha, \pi)$, the look-ahead state $\delta(\alpha)$ must appear in the left-hand side of a rule. The notion of 'deterministic' is also defined as for an original L-PDTA.

Theorem 1: A real-time deterministic L-PDTA, a deterministic L-PDTA and a deterministic L-PDTA with regular look-ahead can recognize an input tree in linear time of the number of nodes of the input tree.

Proof. Because each move of an L-PDTA can be done in constant time, any input tree can be recognized by a real-time deterministic L-PDTA in linear time of the number of nodes. Because the number of consecutive ε-moves can be limited by a constant, recognition by a deterministic L-PDTA can be done in linear time. As for a deterministic bottom-up tree automaton, the process of regular look-ahead can also be done in linear time. Therefore, recognition by a deterministic L-PDTA of all three types can be done in linear time. □

By the way, deterministic recognition of trees recognized by a nondeterministic L-PDTA can be done in $O(n^3)$ time ([8] with some modification).

4. Comparison of Deterministic and Nondeterministic Linear Pushdown Tree Automata

In this section, we compare the recognition capability of the three types of deterministic L-PDTAs and nondeterministic L-PDTAs with regard to tree languages and yield languages.

First, we compare the classes of tree languages. Let **RtDet**, **Det**, **DetRla** and **Nondet** represent classes of all tree languages recognized by real-time deterministic L-PDTAs, deterministic L-PDTAs, deterministic L-PDTAs with regular look-ahead and nondeterministic L-PDTAs, respectively. Then we obtain the following theorem:

Theorem 2: **RtDet** $\subsetneq$ **Det** $\subsetneq$ **DetRla** $\subsetneq$ **Nondet**.

Proof. The inclusions **RtDet** ⊆ **Det** ⊆ **DetRla** are direct from the definitions. The inclusion **DetRla** ⊆ **Nondet** holds because a nondeterministic L-PDTA can simulate a deterministic L-PDTA with regular look-ahead by nondeterministically guessing look-ahead states assigned to the nodes of an input tree in the look-ahead stage.

For the inequality **RtDet** ≠ **Det**, we think of the LM-CFTG $\mathcal{G} = (\{S, A, B\}, \Sigma, P, S)$ where $\Sigma = \Sigma_0 \cup \Sigma_1$, $\Sigma_0 = \{\#\}$, $\Sigma_1 = \{a, b, c, d\}$, and P consists of the following rules: $S \to A(\#)$, $S \to B(\#)$, $A(x) \to a(A(a(x)))$, $A(x) \to b(A(b(x)))$, $A(x) \to a(c(a(x)))$, $A(x) \to b(c(b(x)))$, $B(x) \to a(B(a(x)))$, $B(x) \to b(B(x))$, $B(x) \to a(d(a(x)))$ and $B(x) \to b(d(x))$. Clearly, $L(\mathcal{G}) \in$ **Det** but $L(\mathcal{G}) \notin$ **RtDet**.

The inequality **Det** ≠ **DetRla** is because **DetRla** includes all recognizable tree languages but **Det** doesn't. It is known that the recognizable tree language $\{f(a, b), f(b, a)\}$ is not in **Det**. **DetRla** includes all recognizable tree languages because deterministic L-PDTAs with regular look-ahead can simulate all bottom-up tree automata.

For the inequality **DetRla** ≠ **Nondet**, we think of the LM-CFTG $\mathcal{G} = (\{S, A\}, \Sigma, P, S)$ where $\Sigma = \Sigma_0 \cup \Sigma_1$, $\Sigma_0 = \{\#\}$, $\Sigma_1 = \{a, b\}$, and P consists of the following rules: $S \to A(\#)$, $A(x) \to a(A(a(x)))$, $A(x) \to b(A(b(x)))$, $A(x) \to a(a(x))$ and $A(x) \to b(b(x))$. Clearly, $L(\mathcal{G}) \in$ **Nondet** but $L(\mathcal{G}) \notin$ **DetRla**. □

Next, we compare the classes of yield languages. Let **yRtDet**, **yDet**, **yDetRla** and **yNondet** represent the classes of yield languages corresponding to **RtDet**, **Det**, **DetRla** and **Nondet**, respectively.

As the main result, it will be shown that the classes of their yield languages are the same, i.e., the four classes of L-PDTAs mentioned above are all weakly equivalent:

yRtDet = **yDet** = **yDetRla** = **yNondet**.

To reach the main result, we need to prove the equivalence of L-PDTAs and real-time L-PDTAs.

In the string case, it is known that any pushdown automaton can be converted into an equivalent real-time one. This stems from the fact that any CFG can be converted into an equivalent one in Greibach normal form [10]. Similarly, it is also known that any LM-CFTG can be converted into an equivalent one in Greibach-like normal form [7]. So we come across the following lemma:

Lemma 1: For any L-PDTA, we can construct an equivalent real-time L-PDTA.

Proof. Let $M = (Q, \Sigma, \Gamma, q_0, Z_0, R)$ be an L-PDTA. We construct an LM-CFTG $\mathcal{G}$ such that $L(\mathcal{G}) = T(M)$ [4]. It is known that any LM-CFTG can be converted into an equivalent LM-CFTG whose productions are of one of the following forms [7] (See also Fig. 3):

(1) $A \to a$
(2) $A \to b(C_1, \ldots, C_{i-1}, B_1(B_2(\cdots(B_m(D))\cdots)), C_{i+1}, \ldots, C_n)$
(3) $A(x) \to b(C_1, \ldots, C_{i-1}, B_1(B_2(\cdots(B_m(x))\cdots)), C_{i+1}, \ldots, C_n)$

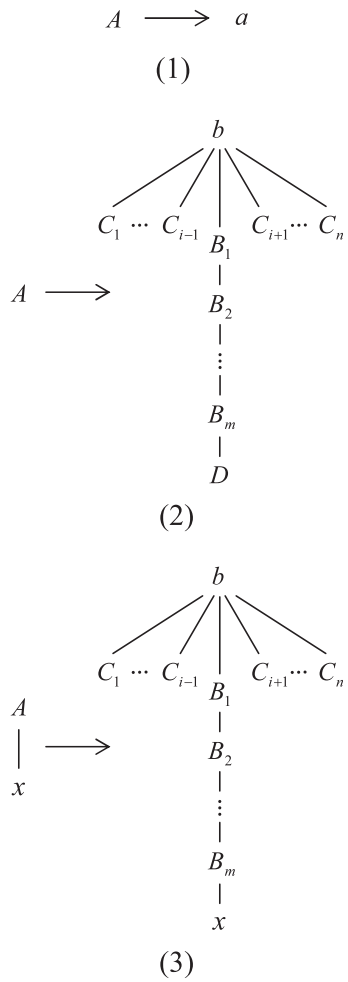

Fig. 3 Rules of an LM-CFTG in Greibach-like normal form.

Here, uppercase characters are nonterminals, lowercase characters are terminals, $n \geq 1$, and $m \geq 0$. Accordingly, $\mathcal{G}$ is converted into $\mathcal{G}'$. Refering to the construction method presented in the proof of Lemma 6.3 in [4], we construct a real-time L-PDTA M' such that $T(M') = L(\mathcal{G}') = T(M)$.

□

Example 2: The following M' is a real-time L-PDTA equivalent to the L-PDTA M introduced in Example 1. $M' = (\{q_0\}, \Sigma, \Gamma', q_0, S, R')$, where $\Sigma = \Sigma_0 \cup \Sigma_2$, $\Sigma_0 = \{a, b\}$, $\Sigma_2 = \{d\}$, $\Gamma'_0 = \{S, A, B\}$, $\Gamma'_1 = \{N, C, D, E\}$, and R' consists of the following rules:

r_1: $q_0(d(x_1, x_2), S) \to d(q_0(x_1, A), q_0(x_2, A))$,
r_2: $q_0(d(x_1, x_2), S) \to d(q_0(x_1, B), q_0(x_2, B))$,
r_3: $q_0(d(x_1, x_2), S) \to d(q_0(x_1, A), q_0(x_2, CA))$,
r_4: $q_0(d(x_1, x_2), S) \to d(q_0(x_1, B), q_0(x_2, CB))$,
r_5: $q_0(d(x_1, x_2), C) \to d(q_0(x_1, A), q_0(x_2, D))$,
r_6: $q_0(d(x_1, x_2), C) \to d(q_0(x_1, B), q_0(x_2, E))$,
r_7: $q_0(d(x_1, x_2), C) \to d(q_0(x_1, A), q_0(x_2, CD))$,
r_8: $q_0(d(x_1, x_2), C) \to d(q_0(x_1, B), q_0(x_2, CE))$,
r_9: $q_0(d(x_1, x_2), D) \to d(q_0(x_1, \lambda), q_0(x_2, A))$,
r_{10}: $q_0(d(x_1, x_2), E) \to d(q_0(x_1, \lambda), q_0(x_2, B))$,
r_{11}: $q_0(a, A) \to a$, and

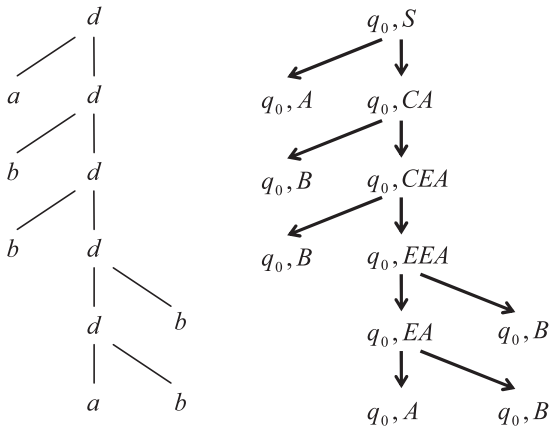

Fig. 4 An example of a tree accepted by M' and its movement.

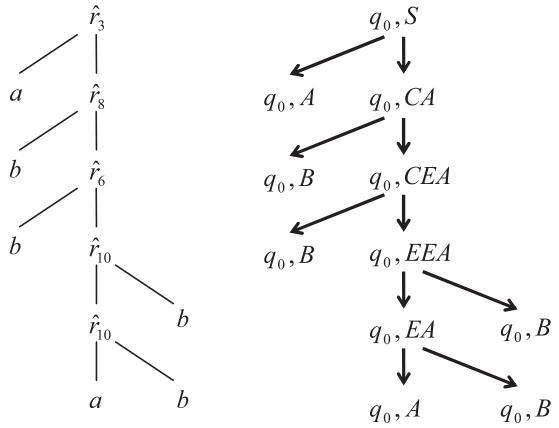

Fig. 5 An example of a tree accepted by M'' and its movement.

r_{12}: $q_0(b, B) \to b$.

Here, $r_1, r_2, \ldots, r_{12}$ are the labels of rules that will be used in the next example. In Fig. 4, a tree in $T(M')$ and the movement of M' for the tree are illustrated.

Some reader may want to ask whether the similar equivalence holds between PDTAs and real-time PDTAs. The answer is negative because the following tree language $L_{b(\alpha,\alpha)}$ is recognized by a PDTA but not by any real-time PDTA. $L_{b(\alpha,\alpha)} = \{b(\alpha, \alpha) \mid \alpha = a(a(\cdots(a(\varepsilon))\cdots))$ is a tree in $T_{\Sigma_0 \cup \Sigma_1}\}$, where $\Sigma = \{\varepsilon, a, b\}$, $\Sigma_0 = \{\varepsilon\}$, $\Sigma_1 = \{a\}$, and $\Sigma_2 = \{b\}$.

Theorem 3: For any L-PDTA, we can construct a weakly equivalent real-time deterministic L-PDTA.

Proof. Let $M = (Q, \Sigma, \Gamma, q_0, Z_0, R)$ be an L-PDTA. By Lemma 1, without loss of generality, we may assume that M is real-time.

For $r \in R$, let us denote the input symbol occurring in r by $\sigma(r)$. Then we construct an L-PDTA $M' = (Q, \Sigma', \Gamma, q_0, Z_0, R')$ as follows. $\Sigma' = \Sigma_0 \cup \{\hat{r} \mid r \in R$ and $\sigma(r) \in \Sigma - \Sigma_0\}$, where the arity of $\hat{r} \in R$ is arity$(\sigma(r))$. R' is the smallest set satisfying the following conditions:

- If $r \in R$, and $\sigma(r) \in \Sigma_0$, then r is in R'.
- If $r \in R$, and $\sigma(r) \in \Sigma - \Sigma_0$, then the rule obtained from r by replacing each occurrence of $\sigma(r)$ with $\hat{r}$ is in R'.

For example, if the rule

$$r: \ q(b(x_1, x_2), A) \to b(q_1(x_1, \pi_1), q_2(x_2, \pi_2))$$

is in R, then the following rule is contained in R':

$$q(\hat{r}(x_1, x_2), A) \to \hat{r}(q_1(x_1, \pi_1), q_2(x_2, \pi_2)).$$

It is easy to check that M' is real-time and deterministic.

By the construction of M', there is one to one correspondence between the move of M and the move of M'. Thus, for any $\alpha \in T_\Sigma$, $q_0(Z_0, \alpha) \overset{*}{\underset{M}{\vdash}} \alpha$ if and only if there exists $\alpha' \in T_{\Sigma'}$ such that $q_0(Z_0, \alpha') \overset{*}{\underset{M'}{\vdash}} \alpha'$, and α is obtained from α' by replacing each occurrence of $\hat{r} \in \Sigma' - \Sigma_0$ with $\sigma(r)$. Therefore, M' is weakly equivalent to M. □

Corollary 1: The following classes of L-PDTAs are all weakly equivalent.

- Real-time deterministic L-PDTAs
- Deterministic L-PDTAs
- Deterministic L-PDTAs with regular look-ahead
- Nondeterministic L-PDTAs

Example 3: The following M'' is a real-time deterministic L-PDTA weakly equivalent to the L-PDTAs M and M' introduced in Example 1 and 2. M'' is obtained from M' in accordance with the construction method presented in the proof of Theorem 3. $M'' = (\{q_0\}, \Sigma', \Gamma', q_0, S, R'')$, where $\Sigma = \Sigma_0 \cup \Sigma_2$, $\Sigma_0 = \{a, b\}$, $\Sigma_2 = \{\hat{r}_1, \hat{r}_2, \ldots, \hat{r}_{10}\}$, $\Gamma'_0 = \{S, A, B\}$, $\Gamma'_1 = \{N, C, D, E\}$, and R'' consists of the following rules:

$q_0(\hat{r}_1(x_1, x_2), S) \to \hat{r}_1(q_0(x_1, A), q_0(x_2, A))$,
$q_0(\hat{r}_2(x_1, x_2), S) \to \hat{r}_2(q_0(x_1, B), q_0(x_2, B))$,
$q_0(\hat{r}_3(x_1, x_2), S) \to \hat{r}_3(q_0(x_1, A), q_0(x_2, CA))$,
$q_0(\hat{r}_4(x_1, x_2), S) \to \hat{r}_4(q_0(x_1, B), q_0(x_2, CB))$,
$q_0(\hat{r}_5(x_1, x_2), C) \to \hat{r}_5(q_0(x_1, A), q_0(x_2, D))$,
$q_0(\hat{r}_6(x_1, x_2), C) \to \hat{r}_6(q_0(x_1, B), q_0(x_2, E))$,
$q_0(\hat{r}_7(x_1, x_2), C) \to \hat{r}_7(q_0(x_1, A), q_0(x_2, CD))$,
$q_0(\hat{r}_8(x_1, x_2), C) \to \hat{r}_8(q_0(x_1, B), q_0(x_2, CE))$,
$q_0(\hat{r}_9(x_1, x_2), D) \to \hat{r}_9(q_0(x_1, \lambda), q_0(x_2, A))$,
$q_0(\hat{r}_{10}(x_1, x_2), E) \to \hat{r}_{10}(q_0(x_1, \lambda), q_0(x_2, B))$,
$q_0(a, A) \to a$, and
$q_0(b, B) \to b$.

In Fig. 5, a tree in $T(M'')$ and the movement of M'' for the tree are illustrated.

5. Conclusion

We have studied the following three types of deterministic L-PDTAs: (1) real-time deterministic L-PDTAs, (2) deterministic L-PDTAs, and (3) deterministic L-PDTAs with regular look-ahead. Concerning tree languages, we have seen that a strict hierarchy holds among the classes of recognized tree languages. Concerning yield languages, on the other hand, we have seen that the classes of their yield languages are the same.

It is recommended to use deterministic L-PDTAs with regular look-ahead for application because they are most powerful among the three types of deterministic L-PDTAs, and their recognition process can be done in linear time.

References

[1] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (XML) 1.0 (fourth edition) W3C recommendation," Available on: http://www.w3.org/TR/2006/REC-xml-20060816/, 2006.

[2] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi, "Tree automata techniques and applications," Available on: http://www.grappa.univ-lille3.fr/tata, 2007. release Oct. 12th 2007.

[3] J. Engelfriet, "Top-down tree transducers with regular look-ahead," Mathematical Systems Theory, vol.10, no.1, pp.289–303, 1977.

[4] A. Fujiyoshi and T. Kasai, "Spinal-formed context-free tree grammars," Theory of Computing Systems, vol.33, no.1, pp.59–83, 2000.

[5] A. Fujiyoshi, "Linearity and nondeletion on monadic context-free tree grammars," Inf. Process. Lett., vol.93, no.3, pp.103–107, 2005.

[6] A. Fujiyoshi and I. Kawaharada, "Deterministic recognition of trees accepted by a linear pushdown tree automaton," Proc. 10th Conference on Implementation and Application of Automata (CIAA 2005), LNCS 3845, pp.129–140, 2005.

[7] A. Fujiyoshi, "Analogical conception of Chomsky normal form and Greibach normal form for linear, monadic context-free tree grammars," IEICE Trans. Inf. & Syst., vol.E89-D, no.12, pp.2933–2938, Dec. 2006.

[8] A. Fujiyoshi, "Application of the CKY algorithm to recognition of tree structures for linear, monadic context-free tree grammars," IEICE Trans. Inf. & Syst., vol.E90-D, no.2, pp.388–394, Feb. 2007.

[9] I. Guessarian, "Pushdown tree automata," Mathematical Systems Theory, vol.16, no.4, pp.237–263, 1983.

[10] J.E. Hopcroft and J.D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison Wesley, Reading, Massachusetts, 1979.

[11] A.K. Joshi and Y. Schabes, Handbook of Formal Languages, ch. Tree-adjoining grammars, pp.69–124, Springer-Verlag, Berlin, 1997.

[12] M. Murata, D. Lee, M. Mani, and K. Kawaguchi, "Taxonomy of XML schema languages using formal language theory," ACM Trans. Internet Technology, vol.5, no.4, pp.660–704, 2005.

[13] W.C. Rounds, "Mapping and grammars on trees," Mathematical Systems Theory, vol.4, no.3, pp.257–287, 1970.

Akio Fujiyoshi was born in Tokyo, Japan in 1971. He received the B.E., M.E., and Dr. Sci. degrees from the University of Electro-Communications, Tokyo, Japan, in 1995, 1997, and 2000, respectively. He is presently an assistant professor in the Department of Computer and Information Sciences, Ibaraki University. His main interests are formal language theory and algorithmic learning theory.