Acta Cybernetica **18** (2008) 427–450.

Pebble Alternating Tree-Walking Automata and Their Recognizing Power*

Loránd Muzamel†

Abstract

Pebble tree-walking automata with alternation were first investigated by Milo, Suciu and Vianu (2003), who showed that tree languages recognized by these devices are exactly the regular tree languages. We strengthen this by proving the same result for pebble automata with "strong pebble handling" which means that pebbles can be lifted independently of the position of the reading head and without moving the reading head. Then we make a comparison among some restricted versions of these automata. We will show that the deterministic and non-looping pebble alternating tree-walking automata are strictly less powerful than their nondeterministic counterparts, i.e., they do not recognize all the regular tree languages. Moreover, there is a proper hierarchy of recognizing capacity of deterministic and non-looping n-pebble alternating tree-walking automata with respect to the number of pebbles, i.e., for each $n \geq 0$, deterministic and non-looping $(n+1)$-pebble alternating tree-walking automata are more powerful than their n-pebble counterparts.

1 Introduction

The concept of a *tree-walking automaton (twa)* was introduced in [1] for modeling the syntax-directed translation from strings to strings. A twa A, obeying its state-behaviour, walks on the edges of the input tree s and accepts s if the (only) accepting state q_{yes} is accessed. Every tree language recognized by a twa is regular. It was an open problem for more than 30 years whether twa can be determinized or whether twa can recognize all regular tree languages. The answer for these two questions were provided in [4] and [3] saying that (1) twa cannot be determinized and (2) twa do not recognize all regular tree languages. Hence $dTWA \subset TWA \subset REG$, where $dTWA$ and TWA denote the tree language classes recognized by deterministic twa and twa, respectively, and REG is the class of regular tree languages.

*Research of the author was partially supported by German Research Foundation (DFG) under GrantGK 334/3 during his stay in the period February-April 2005 at TU Dresden, and also was supported by the Hungarian Scientific Foundation (OTKA) under Grant T 030084.

†Department of Computer Science, University of Szeged, Árpád tér 2., H-6720 Szeged, Hungary, E-mail: muzamel@inf.u-szeged.hu

The generalization of twa with nested pebbles came recently, by two independent motivations: On the one hand, with the advancement of XML theory, finite state recognizers (with name *n-pebble tree automata*) were used in [21] to show that the XML typechecking problem is decidable. On the other hand, the concept of *n-pebble tree-walking automata (n-ptwa)* were defined in [9] to recognize first-order logic on trees. Later, in [10] n-ptwa were extended with a more general pebble handling. In the present paper we will consider tree recognizers along the line of [10].

An n-ptwa A is equipped with a pointer (or reading head), and n different pebbles, which are denoted by $1, \ldots, n$. The pointer of A walks on the edges of an input tree s, while the pebbles can be dropped at and lifted from a node of s in a *stack-like fashion* which means the following:

Dropping of pebbles: If there are $l < n$ pebbles on s, then pebble $l + 1$ can be dropped at the node pointed by the pointer.

Lifting of pebbles: There are two different approaches.

> *weak pebble handling:* If there are $l > 0$ pebbles on s, then pebble l can be lifted iff it is placed at the node pointed by the pointer.
>
> *strong pebble handling:* If there are $l > 0$ pebbles on s, then pebble l can be lifted independently of the position of the pointer.

The automaton A computes on s as follows. Initially, A is in the initial state q_0, its pointer points to the root of s, and no pebbles are placed on s. Then – applying its rules – A moves along the edges of the input tree, drops, and lifts pebbles in a stack-like fashion (with strong or weak pebble handling, depending on the definition). Each step depends on (1) the current state, (2) the presence of the pebbles on the input tree, and (3) the position of the pointer. *A accepts s*, if the (only) accepting state q_{yes} is accessed. Otherwise, *A rejects s*. We say that L is the tree language *recognized by A*, if L contains exactly the trees accepted by A.

Originally, the n-ptwa was defined in [9] with weak pebble handling. In the the present paper we are interested in the more general strong pebble handling, which was used in [10, 22, 5].

In [10] it was proved that tree languages recognized by ptwa are regular. In [5] it was shown that there is a proper hierarchy of the recognizing power of n-ptwa with respect to n, moreover, there is a regular tree language which cannot be recognized by any ptwa. Formally,

$$TWA \subset 1\text{-}PTWA \subset 2\text{-}PTWA \subset \ldots \subset PTWA \subset REG,$$

where n-$PTWA$ denotes the class of tree languages recognized by n-ptwa, for $n \geq 0$, and $PTWA = \bigcup_{n\geq 0} n\text{-}PTWA$.

It was also an interesting and surprising result of [5] that ptwa with strong pebble handling have the same recognizing power as those with weak pebble handling.

Alternation was introduced in [7] as a natural generalization of nondeterminism for Turing machines, finite automata, and pushdown automata. Due to the generality of the concept, it is obvious how to define alternation for other types of sequential automata. For various kinds of (sequential) tree automata, alternation was first investigated in [23]. In [10] it was left as an open problem, whether the tree languages recognized by n-ptwa with alternation and strong pebble handling are regular or not.

In the remainder of this paper we will consider pebble tree-walking devices only with strong pebble handling. Moreover, the definition of n-patwa in the present paper will follow the line of the definition of n-pebble tree transducers of [11].

A computation of an *n-pebble alternating tree-walking automaton (n-patwa)* A on an input tree s starts in the initial state with the pointer at the root node, and there are no pebbles on s. Depending on the applicable rules it generates new parallel computations (such that each has its own copy of s with the current position of the pointer, and the pebbles). The automaton A accepts s if all the computations spawned from the initial configuration terminate in the (only) accepting state q_{yes}. We say that L is the tree language *recognized by* A if L contains exactly the trees accepted by A. In case $n = 0$, we write *alternating tree-walking automaton (atwa)* for 0-patwa. We denote the tree language class recognized by n-patwa, deterministic n-patwa, atwa, and deterministic atwa by *n-PATWA*, *n-dPATWA*, *ATWA*, and *dATWA*, respectively. The unions $\bigcup_{n\geq 0} n\text{-}PATWA$ and $\bigcup_{n\geq 0} n\text{-}dPATWA$ are denoted by *PATWA* and *dPATWA*, respectively.

As main result of this paper, we answer the open problem raised at page 18 of [10] and prove that for all $n \geq 0$, n-patwa recognize exactly the regular tree languages, i.e., $n\text{-}PATWA = REG$.

Roughly speaking, an n-patwa A is *looping* if there is an input tree s such that one of the computations of A on s gets into an infinite loop. Otherwise A is *non-looping*. We denote the non-looping version of the above tree language classes by subscripting an '*nl*' to their names e.g. $dTWA_{nl}$, $dATWA_{nl}$, $n\text{-}dPATWA_{nl}$, etc.

In the second part of this paper we investigate the recognizing power of deterministic non-looping subclasses of the above tree language classes and show that these subclasses do not recognize all the regular tree languages, moreover the following proper inclusion hierarchy holds:

$dTWA \subset dATWA_{nl} \subset 1\text{-}dPATWA_{nl} \subset 2\text{-}dPATWA_{nl} \ldots \subset dPATWA_{nl} \subset REG.$ $(*)$

The paper is organized as follows. In Section 2 we define the necessary concepts. In Section 3 we give the formal definition of an n-patwa and define the looping property for them. In Section 4 we present our main result and prove that n-patwa recognize the regular tree languages. In Section 5 we prove the proper hierarchy $(*)$. Finally, in Section 6 we conclude our results and give some future research topics.

2 Preliminaries

2.1 Sets, strings, and relations

We denote the set of nonnegative integers by $\mathbb{N}$. For every $n \in \mathbb{N}$, we let $[n] = \{1, \ldots, n\}$.

For a set A, $\mathcal{P}(A)$ denotes the power set of A. The empty set is denoted by $\emptyset$. If it does not lead to confusion, we write a for a singleton set $\{a\}$.

For a set A, A^* denotes the set of *strings* (or: *words*) over A; the *empty string* is denoted by ε. For a string $w \in A^*$, $|w|$ denotes its *length*. For every $n \geq 0$, we define $A^{\leq n} = \{u \in A^* \mid |u| \leq n\}$. For every $u \in A^*$, and $1 \leq l \leq |u|$, $u(l)$ denotes the l-th element of A in u.

An *alphabet* is a finite nonempty set. Let A be an alphabet and $L \subseteq A^*$ a finite, nonempty set. We write the strings of L^* in the form $[u_1; \ldots; u_l]$, where $l \geq 0$ and $u_1, \ldots, u_l \in L$. The empty string over L is denoted by $[\,]$.

Let $\rho \subseteq H \times H$ be a binary relation. The fact that $(a, b) \in \rho$ for some $a, b \in H$ is also denoted by $a \,\rho\, b$. For every $l \geq 0$, the l-th power of ρ is denoted by ρ^l, the transitive closure, and the reflexive, transitive closure of ρ are denoted by ρ^+ and ρ^*, respectively.

2.2 Trees and tree languages

A *ranked set* is an ordered pair $(\Sigma, \mathit{rank}_\Sigma)$, where Σ is a set and rank_Σ is a mapping of type $\Sigma \to \mathbb{N}$. If Σ is an alphabet, then $(\Sigma, \mathit{rank}_\Sigma)$ is a *ranked alphabet*. If $\mathit{rank}_\Sigma(\sigma) = k$ for $\sigma \in \Sigma$ and $k \geq 0$, then the rank of σ is k and we indicate this fact also by writing $\sigma^{(k)}$. For every $k \geq 0$, we define $\Sigma^{(k)} = \{\sigma \in \Sigma \mid \mathit{rank}_\Sigma(\sigma) = k\}$. If Σ is clear from the context, we write rank instead of rank_Σ, moreover, we drop rank_Σ and write a ranked set as Σ.

We denote by $\mathit{maxrank}(\Sigma)$ the maximum of ranks of symbols of Σ, i.e., $\mathit{maxrank}(\Sigma) = \max\{\mathit{rank}(\sigma) \mid \sigma \in \Sigma\}$.

Let Σ be a ranked set. The set of *trees over* Σ, denoted by T_Σ, is the smallest set of strings $T \subseteq (\Sigma \cup \{(,)\} \cup \{,\})^*$ such that $\Sigma^{(0)} \subseteq T$ and whenever $k \geq 1$, $\sigma \in \Sigma^{(k)}$, and $t_1, \ldots, t_k \in T$, then $\sigma(t_1, \ldots, t_k) \in T$. Certainly, $T_\Sigma \neq \emptyset$ if and only if $\Sigma^{(0)} \neq \emptyset$.

For every tree $s \in T_\Sigma$, we define the set $\mathit{pos}(s) \subseteq [\mathit{maxrank}(\Sigma)]^*$ of *the nodes of* s as follows. We let $\mathit{pos}(s) = \{\varepsilon\}$ if $s \in \Sigma^{(0)}$, and $\mathit{pos}(s) = \{iu \mid 1 \leq i \leq k,\ u \in \mathit{pos}(s_i)\}$ if $s = \sigma(s_1, \ldots, s_k)$ for some $k \geq 1$, $\sigma \in \Sigma^{(k)}$ and $s_1, \ldots, s_k \in T_\Sigma$.

Now, for a tree $s \in T_\Sigma$ and a node $u \in \mathit{pos}(s)$, we define $\mathit{lab}(s, u) \in \Sigma$, i.e., *the label of* s *at node* u, by induction:

(i) if $s \in \Sigma^{(0)}$ (which implies $u = \varepsilon$), then $\mathit{lab}(s, u) = s$;

(ii) if $s = \sigma(s_1, \ldots, s_k)$ for some $k \geq 1$, $\sigma \in \Sigma^{(k)}$ and trees $s_1, \ldots, s_k \in T_\Sigma$, then

   - if $u = \varepsilon$, then $\mathit{lab}(s, u) = \sigma$,
   - if $u = iu'$, where $1 \leq i \leq k$, and $u' \in \mathit{pos}(s_i)$, then $\mathit{lab}(s, u) = \mathit{lab}(s_i, u')$.

For every $s \in T_\Sigma$ and $u \in \mathit{pos}(s)$ we define the parent of u, denoted by $\mathit{parent}(u)$ and the child number of u, denoted by $\mathit{childno}(u)$ as follows:

(i) if $u = \varepsilon$, then $\mathit{childno}(u) = 0$ and $\mathit{parent}(u)$ is undefined,

(ii) if $u = u'j$ for some $u' \in \mathit{pos}(s)$ and $j \in \mathbb{N}$, then $\mathit{childno}(u) = j$ and $\mathit{parent}(u) = u'$.

If Σ is a ranked alphabet, then any subset $L \subseteq T_\Sigma$ is a *tree language.* The *complement of* L is the tree language $\overline{L} = T_\Sigma - L$. If $\mathcal{L}$ is a class of tree languages, then $co\text{-}\mathcal{L} = \{\overline{L} \mid L \in \mathcal{L}\}$.

We will need a tree recognizer concept called *top-down tree automaton.* The unfamiliar reader can consult with [17, 18] for this concept, although they called it *root-to-frontier automaton.* A tree language is *regular*, if it can be recognized by a top-down tree automaton. We denote the class of regular tree languages by REG. The following classical result saying that regular tree languages are closed under complementation will be needed later.

Proposition 2.1 $\mathit{REG} = co\text{-}\mathit{REG}$.

2.3 MSO logic for trees

Monadic second order (MSO) logic was originally proposed to describe properties of strings in [6]. MSO logic can be extended for trees, see [24, 8, 2]. We will recall the syntax and the semantics of this logic over a ranked alphabet Σ.

Syntax:

We define the language $\mathit{MSOL}(\Sigma)$ of *MSO formulas (over* Σ*).* This language is built up from the following symbols.

node variables: $x, y, x_1, x_2, \ldots$. We denote the set of node variables by VAR_1.

node-set variables: $X, X_1, X_2, \ldots$. We denote the set of node-set variables by VAR_2.

other symbols: $\neg$, $\wedge$, $\exists$, (,)

Atomic formulas are strings of one of the following types:

- $\mathit{lab}_\sigma(x)$, where $\sigma \in \Sigma$, and $x \in \mathit{VAR}_1$,
- $\mathit{child}_i(x_1, x_2)$, where $1 \leq i \leq \mathit{maxrank}(\Sigma)$, and $x_1, x_2 \in \mathit{VAR}_1$,
- $x \in X$, where $x \in \mathit{VAR}_1$, and $X \in \mathit{VAR}_2$.

The language of *MSO formulas over* Σ is the smallest set $\mathit{MSOL}(\Sigma)$ satisfying the following conditions.

(i) Each atomic formula is a formula of $\mathit{MSOL}(\Sigma)$.

(ii) Let $\phi_1, \phi_2 \in \mathit{MSOL}(\Sigma)$, $x \in \mathit{VAR}_1$, and $X \in \mathit{VAR}_2$. Then $(\neg\phi_1)$, $(\phi_1 \wedge \phi_2)$, $\exists x(\phi_1)$, $\exists X(\phi_1) \in \mathit{MSOL}(\Sigma)$.

Let $\phi \in \mathit{MSOL}(\Sigma)$ be an MSO formula and x (X) a node (node-set) variable in ϕ. Then an occurrence of x (X) in ϕ is said to be *free* in ϕ, if x (X) is not in the scope of $\exists x$ $(\exists X)$, otherwise that occurrence is *bound* in ϕ. The formulas without free occurrences of node and node-set variables are the *closed formulas.*

Semantics:

The truth value of a formula is considered through structures. A *structure (over Σ)* is a triple (s, Π_1, Π_2), where

- $s \in T_\Sigma$,
- $\Pi_1 : \mathit{VAR}_1 \to \mathit{pos}(s)$, and
- $\Pi_2 : \mathit{VAR}_2 \to \mathcal{P}(\mathit{pos}(s))$.

Now, let (s, Π_1, Π_2) be a structure and $\phi \in \mathit{MSOL}(\Sigma)$ a formula. We define that the structure (s, Π_1, Π_2) *models* $\phi \in \mathit{MSOL}(\Sigma)$, or ϕ *is true in* (s, Π_1, Π_2) (denoted by $(s, \Pi_1, \Pi_2) \models \phi$) by formula induction on ϕ as follows.

(i)/a If $\phi = \mathit{lab}_\sigma(x)$, then $(s, \Pi_1, \Pi_2) \models \phi$ iff the label of the node $\Pi_1(x)$ is σ.

(i)/b If $\phi = \mathit{child}_i(x_1, x_2)$, then $(s, \Pi_1, \Pi_2) \models \phi$ iff node $\Pi_1(x_2)$ is the parent of node of $\Pi_1(x_1)$ and $\mathit{childno}(\Pi_1(x_1)) = i$.

(i)/c If $\phi = x \in X$, then $(s, \Pi_1, \Pi_2) \models \phi$ iff $\Pi_1(x) \in \Pi_2(X)$.

(ii)/a If $\phi = (\neg\phi_1)$, then $(s, \Pi_1, \Pi_2) \models \phi$ iff $(s, \Pi_1, \Pi_2) \not\models \phi_1$.

(ii)/b If $\phi = (\phi_1 \wedge \phi_2)$, then $(s, \Pi_1, \Pi_2) \models \phi$ iff $(s, \Pi_1, \Pi_2) \models \phi_1$, and $(s, \Pi_1, \Pi_2) \models \phi_2$.

(ii)/c If $\phi = \exists x(\phi_1)$, then $(s, \Pi_1, \Pi_2) \models \phi$ iff there is a node $u \in \mathit{pos}(s)$ and a structure (s, Π_1', Π_2), such that for every $y \in \mathit{VAR}_1$, we have

$$\Pi_1'(y) = \begin{cases} u & \text{if } y = x, \\ \Pi_1(y) & \text{if } y \neq x, \end{cases}$$

and $(s, \Pi_1', \Pi_2) \models \phi_1$.

(ii)/d If $\phi = \exists X(\phi_1)$, then $(s, \Pi_1, \Pi_2) \models \phi$ iff there is a node set $U \subseteq \mathit{pos}(s)$ and a structure (s, Π_1, Π_2'), such that for every $Y \in \mathit{VAR}_2$, we have

$$\Pi_2'(Y) = \begin{cases} U & \text{if } Y = X, \\ \Pi_2(Y) & \text{if } Y \neq X, \end{cases}$$

and $(s, \Pi_1, \Pi_2') \models \phi_1$.

To improve the readability of a formula, we omit the outer brackets. Moreover, we will use the standard shorthand $\phi_1 \vee \phi_2$ for $\neg\phi_1 \wedge \neg\phi_2$, $\phi_1 \to \phi_2$ for $\neg\phi_1 \vee \phi_2$, $\forall x\phi$ for $\neg\exists x \neg\phi$, and $\forall X\phi$ for $\neg\exists X\neg\phi$.

It is straightforward that for a closed formula $\phi \in \mathit{MSOL}(\Sigma)$, and structure (s, Π_1, Π_2), the mappings Π_1 and Π_2 do not influence the fact that $(s, \Pi_1, \Pi_2) \models \phi$ or not. Hence for a closed formula ϕ, we will write $s \models \phi$ for $(s, \Pi_1, \Pi_2) \models \phi$.

Let $\phi \in \mathit{MSOL}(\Sigma)$ be a closed formula. The tree language *defined by* ϕ is the tree language $L(\phi) = \{s \in T_\Sigma \mid s \models \phi\}$. A tree language $L \subseteq T_\Sigma$ is *MSO-definable*, if there is a closed formula $\phi \in \mathit{MSOL}(\Sigma)$, where $L = L(\phi)$. The following classical result from [8, 24] states that the MSO-definable tree languages are exactly the regular tree languages.

Proposition 2.2 A tree language is MSO-definable if and only if it is regular. $\diamond$

3 Pebble alternating tree-walking automata

3.1 Syntax and semantics

In this section we introduce the concept of an *n-pebble alternating tree-walking automaton*. For this, we define the set of *instructions*.

Definition 3.1 For every integer $d \geq 0$, let

$$I_d = \{\mathit{stay}, \mathit{up}, \mathit{drop}, \mathit{lift}, \mathit{down}_1, \mathit{down}_2, \ldots, \mathit{down}_d\}.$$

The elements of I_d are called *instructions*.

For a ranked alphabet Σ, symbol $\sigma \in \Sigma$, $n \geq 0$, bit vector $b \in \{0,1\}^{\leq n}$, and $j \in \{0, 1, \ldots, \mathit{maxrank}(\Sigma)\}$, let $I_{\sigma,b,j,n} \subseteq I_{\mathit{rank}(\sigma)}$ be the smallest set satisfying the following conditions:

(i) $\mathit{stay} \in I_{\sigma,b,j,n}$,

(ii) if $j \neq 0$, then $\mathit{up} \in I_{\sigma,b,j,n}$,

(iii) for every $1 \leq i \leq \mathit{rank}(\sigma)$ we have $\mathit{down}_i \in I_{\sigma,b,j,n}$,

(iv) if $|b| < n$, then $\mathit{drop} \in I_{\sigma,b,j,n}$,

(v) if $b \neq \varepsilon$, then $\mathit{lift} \in I_{\sigma,b,j,n}$.

If n is clear from the context, then we write $I_{\sigma,b,j}$ for $I_{\sigma,b,j,n}$. $\diamond$

Definition 3.2 For $n \geq 0$, an *n-pebble alternating tree-walking automaton* (shortly *n-patwa*) is a system $A = (Q, \Sigma, q_0, q_{yes}, R)$, where

- Q is a finite nonempty set, the *set of states*, which is partitioned into pairwise disjoint subsets $Q_0, Q_1, \ldots, Q_n$,
- Σ is a ranked alphabet, the *input alphabet*,
- $q_0 \in Q_0$ is a distinguished state, the *initial state*,
- $q_{yes} \notin Q$ is a new state, the *accepting state*,
- R is a finite *set of rules*, which is partitioned into pairwise disjoint subsets $R_0, R_1, \ldots, R_n$, such that for each $0 \leq i \leq n$, the set R_i consists of
  - *accepting rules* of the form $\langle q, \sigma, b, j\rangle \to \langle q_{yes}, \mathit{stay}\rangle$,
  - *pebble tree-walking rules* of the form $\langle q, \sigma, b, j\rangle \to \langle p, \varphi\rangle$, and
  - *alternating rules* of the form $\langle q, \sigma, b, j\rangle \to \{\langle p_1, \mathit{stay}\rangle, \langle p_2, \mathit{stay}\rangle\}$,

where $q \in Q_i$, $\sigma \in \Sigma$, $b \in \{0,1\}^i$, $0 \leq j \leq \mathit{maxrank}(\Sigma)$, $p_1, p_2 \in Q_i$, $\varphi \in I_{\sigma,b,j}$, moreover

$$p \in \begin{cases} Q_i & \text{if } \varphi \in \{\mathit{stay}, \mathit{up}, \mathit{down}_1, \mathit{down}_2, \ldots\}, \\ Q_{i+1} & \text{if } \varphi = \mathit{drop}, \text{ and} \\ Q_{i-1} & \text{if } \varphi = \mathit{lift}. \end{cases}$$

◇

By a *pebble alternating tree-walking automaton (patwa)* we mean an n-patwa for some n.

A tree $s \in T_\Sigma$ is called an *input tree to* A or just an input tree. In the remainder of this section A stands for the n-patwa $A = (Q, \Sigma, q_0, q_{yes}, R)$.

We say that A is *deterministic*, if, for every $q \in Q$, $\sigma \in \Sigma$, $b \in \{0,1\}^{\leq n}$, and $j \in \{0, 1, \ldots, \mathit{maxrank}(\Sigma)\}$, there is at most one rule of R with left-hand side $\langle q, \sigma, b, j\rangle$. Next we introduce further syntactic restrictions for patwa.

Definition 3.3 A is

- an *alternating tree-walking automaton* (shortly *atwa*), if A is a 0-patwa.
- an *n-pebble tree-walking automaton* (shortly *n-ptwa*)[10], if there are no alternating rules in R.
- a *tree-walking automaton* (shortly *twa*)[1], if A is a 0-ptwa. ◇

By a *pebble tree-walking automaton (ptwa)* we mean an n-ptwa for some n. Next we make some preparation for defining the semantics of a patwa. First we define the concept of an n-pebble configuration.

Definition 3.4 For an input tree $s \in T_\Sigma$, an *n-pebble configuration* (or: *pebble configuration*) over s (and A) is a pair $h = (u, \pi)$, where $u \in \mathit{pos}(s)$ is a node of s and $\pi \in (\mathit{pos}(s))^{\leq n}$, i.e., π is a string over $\mathit{pos}(s)$ of length at most n. The set of pebble configurations over s and A is denoted by $PC_{A,s}$. ◇

A pebble configuration $h = (u, \pi) \in PC_{A,s}$, with the string of strings $\pi = [u_1; \ldots; u_l]$ contains the information that the node being scanned by A (the current node) of the input tree s is u and A put $l = |\pi|$ pebbles on the nodes $u_1, \ldots, u_l$ of s. Note that more than one pebble can be put on the same node.

We define a mapping that tests a pebble configuration and returns a triple, which will influence the computation relation.

Definition 3.5 Let $s \in T_\Sigma$ be an input tree and $h = (u, \pi) \in PC_{A,s}$ a pebble configuration. Then $\mathit{test}_s(h) = (\sigma, b, j)$, where

- $\sigma = \mathit{lab}(s, u)$,
- $b \in \{0,1\}^*$ is a string (bit vector) of length $l = |\pi|$, where, for every $1 \leq i \leq l$,

  $$b(i) = \begin{cases} 1 & \text{if } \pi(i) = u \\ 0 & \text{if } \pi(i) \neq u, \end{cases}$$

  (Note, it follows from Definition 3.4 that $l \leq n$.)

- $j = \mathit{childno}(u)$. ◇

If s is clear from the context, then we write $\mathit{test}(h)$ for $\mathit{test}_s(h)$. Next we define how an instruction can be executed on a configuration.

Definition 3.6 Let $s \in T_\Sigma$ be an input tree and $h = (u, \pi) \in PC_{A,s}$ an n-pebble configuration over s with $\pi = [u_1; \ldots ; u_l]$. Let $\mathit{test}(h) = (\sigma, b, j)$ and take an instruction $\varphi \in I_{\mathit{test}(h)} = I_{\sigma,b,j}$. The *execution of* φ *on* h is the pebble configuration $\varphi(h)$ defined in the following way.

$$\varphi(h) = \varphi((u, \pi)) = \begin{cases} (u, \pi) & \text{if } \varphi = \mathit{stay}, \\ (\mathit{parent}(u), \pi) & \text{if } \varphi = \mathit{up}, \\ (ui, \pi) & \text{if } \varphi = \mathit{down}_i, \\ (u, [u_1; \ldots ; u_l; u]) & \text{if } \varphi = \mathit{drop}, \\ (u, [u_1; \ldots ; u_{l-1}]) & \text{if } \varphi = \mathit{lift}. \end{cases}$$

◇

Now we define the concept of a *configuration of* A.

Definition 3.7 Let $s \in T_\Sigma$ be an input tree. A *configuration of* A (over s) is a pair $\langle q, h\rangle$, where $q \in Q \cup \{q_{yes}\}$ and $h \in PC_{A,s}$. ◇

Roughly speaking, a configuration is a snapshot of the computation, storing the current state, the node pointed at by the pointer, and the positions of the dropped pebbles. The set of configurations of A over s is denoted by $C_{A,s}$.

Due to alternation, A is capable to do arbitrary many parallel computations (threads) while processing s and hence, the computation relation works over the subsets of $C_{A,s}$. We turn to introduce this computation relation.

Definition 3.8 Let $s \in T_\Sigma$ be an input tree. Then $\vdash_{A,s} \subseteq \mathcal{P}(C_{A,s}) \times \mathcal{P}(C_{A,s})$ is the *computation relation of* A *on* s, where for all configuration sets $H_1, H_2 \in \mathcal{P}(C_{A,s})$ we have $H_1 \vdash_{A,s} H_2$ if and only if there is a configuration $\langle q, h\rangle \in H_1$, such that one of the following is true.

(1) There is an accepting rule $\langle q, \sigma, b, j\rangle \to \langle q_{yes}, \mathit{stay}\rangle$ in R such that $\mathit{test}(h) = (\sigma, b, j)$ and $H_2 = (H_1 - \{\langle q, h\rangle\}) \cup \{\langle q_{yes}, h\rangle\}$.

(2) There is a pebble tree-walking rule $\langle q, \sigma, b, j\rangle \to \langle p, \varphi\rangle$ in R such that $\mathit{test}(h) = (\sigma, b, j)$ and $H_2 = (H_1 - \{\langle q, h\rangle\}) \cup \{\langle p, \varphi(h)\rangle\}$.

(3) There is an alternating rule $\langle q, \sigma, b, j\rangle \to \{\langle p_1, \mathit{stay}\rangle, \langle p_2, \mathit{stay}\rangle\}$ in R such that $\mathit{test}(h) = (\sigma, b, j)$ and $H_2 = (H_1 - \{\langle q, h\rangle\}) \cup \{\langle p_1, h\rangle, \langle p_2, h\rangle\}$. ◇

We note that the role of the alternating rules is to spawn two parallel computations (threads) from one computation, such that the two new computations start out to work from the current pebble configuration. Moreover, each parallel computation has its own copy of the input tree and an own pebble configuration, which cannot be modified by other computations.

Pebble tree-walking rules are responsible for the sequential steps of a computation (moving on the edges, dropping and lifting of pebbles), and accepting rules are for terminating and accepting a computation.

The n-patwa A works as follows on an input tree s. It starts out in the *initial configuration set* $\{\langle q_0, (\varepsilon, [\,])\rangle\}$ (i.e., only one thread, initial state, pointer at the root node, and no pebbles dropped on s). Then, applying $\vdash_{A,s}$ step by step, it computes further configuration sets. The goal is that each parallel computation spawned from the initial configuration should be accepting, in other words, to terminate in a special configuration set $H \in \mathcal{P}(C_{A,s})$, such that the state-component of each configuration of H is q_{yes}. In that case H is an *accepting configuration set.* It is easy to see that there is no computation step from an accepting configuration set.

Let $ACC_{A,s} = \{q_{yes}\} \times PC_{A,s}$ be the largest accepting configuration set. Thus the tree language recognized by A is defined as follows.

Definition 3.9 The *tree language recognized by A* is

$$L(A) = \{s \in T_\Sigma \mid \langle q_0, (\varepsilon, [\,])\rangle \vdash^*_{A,s} H, \text{ for some } H \subseteq ACC_{A,s}\}. \qquad \diamond$$

The classes of tree languages computed by n-patwa, atwa, n-ptwa, and twa are denoted by *n-PATWA*, *ATWA*, *n-PTWA*, and *TWA*, respectively. The unions $\bigcup_{n\geq 0} n\text{-}PATWA$, and $\bigcup_{n\geq 0} n\text{-}PTWA$ are denoted by *PATWA*, and *PTWA*, respectively. The deterministic subclasses of the above tree language classes are denoted by prefixing a letter '*d*' in front of their names, e.g., *n-dPATWA*, *dATWA*.

It should be clear that with the growing number of pebbles, the recognizing power of patwa and ptwa do not decrease, i.e., $n\text{-}PATWA \subseteq (n+1)\text{-}PATWA$, and $n\text{-}PTWA \subseteq (n+1)\text{-}PTWA$ for every $n \geq 0$.

3.2 Looping and non-looping patwa

Now we turn to the looping property of patwa. Roughly speaking, A is looping, if it has an infinite computation on an input tree.

We introduce the looping property for patwa similarly as the circularity concept was introduced for attributed grammars [20, 19], attributed tree transducers [12, 16], and pebble (macro) tree transducers [11, 14, 15].

We say that $\langle q, h\rangle \in C_{A,s}$ is a *looping configuration*, if there is a configuration set $H \subseteq C_{A,s}$, such that $\langle q, h\rangle \in H$ and $\langle q, h\rangle \vdash^+_{A,s} H$. Moreover, A is *looping*, if there is an input tree $s \in T_\Sigma$, a configuration set $H \subseteq C_{A,s}$ such that

- H contains a looping configuration and
- $\langle q_0, (\varepsilon, [\,])\rangle \vdash^*_{A,s} H$.

Otherwise, A is *non-looping.*

The looping property for pebble macro tree transducers appear in [15] by name "strong circularity". Let us denote the non-looping version of the above tree language classes by $n\text{-}PATWA_{nl}$, $n\text{-}dPATWA_{nl}$, etc.

3.3 Patwa with general alternating rules

When using alternation in a patwa, sometimes it is convenient not to be restricted to the forms of the possible right-hand sides of alternating rules of Definition 3.2. It should be clear that we can allow not only two, but arbitrary many state-instruction pairs for the right-hand sides of the alternating rules. Moreover, in the right-hand side of an alternating rule we allow not only *stay*, but arbitrary instructions. A *general alternating rule* is a rule of the form $\langle q, \sigma, b, j\rangle \rightarrow \{\langle q_1, \varphi_1\rangle, \ldots, \langle q_m, \varphi_m\rangle\}$ with $m \geq 1$, $\langle q_1, \varphi_1\rangle, \ldots, \langle q_m, \varphi_m\rangle \in Q \times I_{\sigma,b,j}$. Moreover, we assume that the state set is not partitioned, and $q_1, \ldots, q_m$ can be arbitrary states of $Q \cup \{q_{yes}\}$.

An *n-patwa with general alternating rules* is a tuple $A = (Q, \Sigma, q_0, q_{yes}, R)$, where R is a finite set of general alternating rules (and the rest is as for an n-patwa). For A, the notion 'deterministic', and the concept of 'configuration' are defined in the same way as for an n-patwa.

For defining the computation relation of A we remove point (1) and (2) and modify point (3) in Definition 3.8 in the following way.

(3) There is a general alternating rule $\langle q, \sigma, b, j\rangle \rightarrow \{\langle p_1, \varphi_1\rangle, \ldots, \langle p_m, \varphi_m\rangle\}$ in R such that $test(h) = (\sigma, b, j)$ and $H_2 = (H_1 - \{\langle q, h\rangle\}) \cup \{\langle p_1, \varphi_1(h)\rangle, \ldots, \langle p_m, \varphi_m(h)\rangle\}$.

Finally, the tree language $L(A)$ recognized by A is defined in the same way as in Definition 3.9, and the looping property of A can be defined similarly as in section 3.2.

We leave the proof of the following lemma to the reader.

Lemma 3.10 For every $n \geq 0$, and n-patwa A with general alternating rules, we can construct an n-patwa A', such that

- $L(A) = L(A')$,
- A is deterministic iff A' is deterministic, and
- A is non-looping iff A' is non-looping. ◇

3.4 Some notes about alternation and patwa

The idea of extending Turing machines and automata with alternation comes from [7]. The term "alternation" means the mixture of *existential nondeterminism* and *universal nondeterminism*.

Existential nondeterminism is the classical nondeterminism concept, i.e., a configuration will be accepting, if there is at least one accepting computation which starts out from that configuration. On the other hand, universal nondeterminism means, that a configuration is accepting, if all possible computations which start out of that configuration lead to acceptance.

The mixture of existential and universal nondeterminism is solved in the folklore by partitioning the state set Q into Q_{OR} (existential states), and Q_{AND} (universal states), moreover the configurations with states from Q_{OR} (resp. Q_{AND}) are regarded with existential nondeterminism (resp. universal nondeterminism).

Our definition of patwa differs from the usual alternating devices, because the present form of patwa is sometimes more handable in this paper. In our context, each state is existential. The universal nondeterminism for patwa is due to the alternating rules which spawn parallel computations, such that all of those computations should be accepting in order to accept the input.

However, it is easy to show that the definition of patwa with classical alternation (with existential, universal states and without alternating rules) would yield tree recognizers with the same recognizing power as patwa of the present paper have.

4 The recognizing power of patwa

In this section we show that the tree languages recognized by patwa are exactly the regular tree languages. We closely follow the ideas in the proof of Theorem 4.7 of [21].

It is easy to see that each top-down tree automaton can be simulated by a 0-patwa. To prove the converse, we will give a closed MSO formula ϕ for every patwa A, such that $L(\phi) = L(A)$. Then using Proposition 2.2, we will obtain that each tree language recognized by a patwa is regular. To make the proof more understandable, we will need some abbreviations of $\mathit{MSOL}(\Sigma)$ formulas, which are listed bellow. Let $d = \mathit{maxrank}(\Sigma)$ and $0 \leq i \leq d$ an arbitrary number.

- $\underline{x_1 = x_2} \equiv \forall X(x_1 \in X \leftrightarrow x_2 \in X)$ (that is true in (s, Π_1, Π_2), if node $\Pi_1(x_1)$ equals node $\Pi_1(x_2)$),
- $\underline{\mathit{child}(x_1, x_2)} \equiv \mathit{child}_1(x_1, x_2) \vee \ldots \vee \mathit{child}_d(x_1, x_2)$ (that is true in (s, Π_1, Π_2), if $\Pi_1(x_1)$ is a child of $\Pi_1(x_2)$),
- $\underline{\mathit{root}(x)} \equiv \forall x_1(\neg \mathit{child}(x, x_1))$ (that is true in (s, Π_1, Π_2), if $\Pi_1(x)$ is the root node)
- $\underline{\mathit{root} \in X} \equiv \forall x(\mathit{root}(x) \rightarrow x \in X)$ (that is true in (s, Π_1, Π_2), if the root node is in $\Pi_2(X)$),
- $\underline{\mathit{chno}_i(x)} \equiv \begin{cases} \mathit{root}(x) & \text{if } i = 0 \\ \exists x_1(\mathit{child}_i(x, x_1)) & \text{if } i > 0 \end{cases}$

  (that is true in (s, Π_1, Π_2), if the child number of $\Pi_1(x)$ is i), and
- $\underline{\mathit{true}} \equiv \forall x(x = x)$ (which is a valid formula).

In the remainder of this section let $n \geq 0$, $A = (Q, \Sigma, q_0, q_{yes}, R)$ an n-patwa, and $s \in T_\Sigma$ an input tree to A. We enumerate the states in Q such that $Q =$

$\{q_0,\ldots,q_m\}$, and $Q_0 = \{q_{m_0} = q_0,\ldots,q_{m_1-1}\}, Q_1 = \{q_{m_1},\ldots,q_{m_2-1}\},\ldots,Q_n = \{q_{m_n},\ldots,q_{m_{n+1}-1} = q_m\}$. Let us observe that $m_{n+1} = m+1$.

Next we give an alternative way to define that A accepts or rejects a tree s. In fact, we will define the acceptance through node sets $S_0,\ldots,S_m \subseteq pos(s)$, where for each $0 \le i \le m$, node set S_i is associated with state q_i of A. Note that for $0 \le l \le n$, the node sets concerned with the states of Q_l are $S_{m_l},\ldots,S_{m_{l+1}-1}$, since $Q_l = \{q_{m_l},\ldots,q_{m_{l+1}-1}\}$. Then, we show that the alternative definition of acceptance described below can be expressed by an MSO formula. Hence, by Proposition 2.2, it follows that the tree language accepted by A is regular.

We begin with some preparation. Namely, we define the *closed*, and the *strongly closed* properties for node sets $S_0,\ldots,S_m$.

Definition 4.1 Let $0 \le l \le n$, $u_1,\ldots,u_l \in pos(s)$, and $S_0,\ldots,S_{m_l-1}$, $S_{m_l},\ldots,S_{m_{l+1}-1} \subseteq pos(s)$. We define the node sets $S_{m_l},\ldots,S_{m_{l+1}-1}$ to be *l-closed with respect to A, s, $u_1,\ldots,u_l$, and $S_0,\ldots,S_{m_l-1}$* by a downward induction on l as follows. (Note that the base of the induction is $l = n$.)

(i) If $l = n$, then the following statements hold.

(1) For every $m_l \le \mu \le m_{l+1}-1$ and $u \in pos(s)$, if $\langle q_\mu,(u,[u_1;\ldots;u_l])\rangle \vdash_{A,s} \langle q_{yes},(u,[u_1;\ldots;u_l])\rangle$, then $u \in S_\mu$.

(2) For every $m_l \le \mu,\nu \le m_{l+1}-1$ and $u,u' \in pos(s)$, if $\langle q_\mu,(u,[u_1;\ldots;u_l])\rangle \vdash_{A,s} \langle q_\nu,(u',[u_1;\ldots;u_l])\rangle$, and $u' \in S_\nu$, then $u \in S_\mu$.

(3) For every $m_l \le \mu,\nu_1,\nu_2 \le m_{l+1}-1$ and $u \in pos(s)$, if $\langle q_\mu,(u,[u_1;\ldots;u_l])\rangle \vdash_{A,s} \{\langle q_{\nu_1},(u,[u_1;\ldots;u_l])\rangle,\langle q_{\nu_2},(u,[u_1;\ldots;u_l])\rangle\}$, $u \in S_{\nu_1}$, and $u \in S_{\nu_2}$, then $u \in S_\mu$.

(4) For every $m_l \le \mu \le m_{l+1}-1$, $m_{l-1} \le \nu \le m_l-1$ (provided that $l > 0$), and $u \in pos(s)$, if $\langle q_\mu,(u,[u_1;\ldots;u_l])\rangle \vdash_{A,s} \langle q_\nu,(u,[u_1;\ldots;u_{l-1}])\rangle$ and $u \in S_\nu$, then $u \in S_\mu$.

(ii) Let $l < n$. Then (1)-(4) hold, moreover:

(5) For every $m_l \le \mu \le m_{l+1}-1$, $m_{l+1} \le \nu \le m_{l+2}-1$, and $u \in pos(s)$, if $\langle q_\mu,(u,[u_1;\ldots;u_l])\rangle \vdash_{A,s} \langle q_\nu,(u,[u_1;\ldots;u_l;u])\rangle$, and for all node sets $S_{m_{l+1}},\ldots,S_{m_{l+2}-1}$ that are $(l+1)$-closed with respect to A, s, $u_1,\ldots,u_l,u$, and $S_0,\ldots S_{m_{l+1}-1}$, we have $u \in S_\nu$, then $u \in S_\mu$. ◇

Definition 4.2 Let $0 \le l \le n$, $S_0,\ldots,S_{m_{l+1}-1} \subseteq pos(s)$, and $u_1,\ldots,u_l \in pos(s)$. We say that $S_0,\ldots,S_{m_{l+1}-1}$ are *strongly l-closed with respect to* A, s, and $u_1,\ldots,u_l$, if

$S_0,\ldots,S_{m_1-1}$ are 0-closed with respect to A and s,
$S_{m_1},\ldots,S_{m_2-1}$ are 1-closed with respect to A, s, u_1, and $S_0,\ldots,S_{m_1-1}$,
$\vdots$
$S_{m_l},\ldots,S_{m_{l+1}-1}$ are l-closed with respect to A, s, $u_1,\ldots,u_l$, and $S_0,\ldots,S_{m_l-1}$. ◇

In case $l = 0$ we make the following observation.

Observation 4.3 $S_0, \ldots, S_{m_1-1}$ are strongly 0-closed if and only if $S_0, \ldots, S_{m_1-1}$ are 0-closed with respect to A and s. $\diamond$

Lemma 4.4 Let $u_1, \ldots, u_n \in pos(s)$ be arbitrary nodes. Define the node sets $T_0, \ldots, T_m \subseteq pos(s)$ such that for each $0 \leq l \leq n$, $m_l \leq \mu \leq m_{l+1} - 1$, and $u \in pos(s)$, we have $u \in T_\mu$ iff there is an accepting configuration set $H \subseteq ACC_{A,s}$ such that $\langle q_\mu, (u, [u_1; \ldots; u_l]) \rangle \vdash^+_{A,s} H$.

Then $T_0, \ldots, T_m$ are strongly n-closed with respect to A, s, and $u_1, \ldots, u_n$.

Proof. Let $0 \leq l \leq n$. It suffices to prove that $T_{m_l}, \ldots, T_{m_{l+1}-1}$ are l-closed with respect to A, s, $u_1, \ldots, u_l$, and $T_0, \ldots, T_{m_l-1}$. We prove by induction on l. (Note that the induction base is $l = n$.)

(i) Let $l = n$. It is easy to see that properties (1) - (4) of Definition 4.1 hold for $T_{m_n}, \ldots, T_{m_{n+1}-1}$ with respect to A, s, $u_1, \ldots, u_n$, and $T_0, \ldots, T_{m_n-1}$.

(ii) Let $l < n$. Then, we can also easily see that properties (1) - (4) hold for $T_{m_l}, \ldots, T_{m_{l+1}-1}$ with respect to A, s, $u_1, \ldots, u_l$, and $T_0, \ldots, T_{m_l-1}$. Now we prove that property (5) of Definition 4.1 holds for $T_{m_l}, \ldots, T_{m_{l+1}-1}$ with respect to A, s, $u_1, \ldots, u_l$, and $T_0, \ldots, T_{m_l-1}$ as follows.

Let $m_l \leq \mu \leq m_{l+1} - 1$, $m_{l+1} \leq \nu \leq m_{l+2} - 1$, and $u \in pos(s)$, assume that

(*) $\langle q_\mu, (u, [u_1; \ldots; u_l]) \rangle \vdash_{A,s} \langle q_\nu, (u, [u_1; \ldots; u_l; u]) \rangle$, and

(**) $u \in S_\nu$ for all node sets $S_{m_{l+1}}, \ldots, S_{m_{l+2}-1}$, that are $(l+1)$-closed with respect to A, s, $u_1, \ldots, u_l, u$, and $T_0, \ldots, T_{m_{l+1}-1}$.

Moreover, we define the node sets $T'_{m_{l+1}}, \ldots, T'_{m_{l+2}-1} \subseteq pos(s)$, such that for each $m_{l+1} \leq \eta \leq m_{l+2} - 1$, and $v \in pos(s)$, we have $v \in T'_\eta$ iff there is an accepting configuration set $H \subseteq ACC_{A,s}$ such that $\langle q_\eta, (v, [u_1; \ldots; u_l; u]) \rangle \vdash^+_{A,s} H$.

We make the following observations.

a) By the induction hypothesis, the node sets $T'_{m_{l+1}}, \ldots, T'_{m_{l+2}-1}$ are $(l + 1)$-closed with respect to A, s, $u_1, \ldots, u_l, u$, and $T_0, \ldots, T_{m_{l+1}-1}$.

b) Then, by (**) and a) we obtain that $u \in T'_\nu$.

c) By the definition of $T'_{m_{l+1}}, \ldots, T'_{m_{l+2}-1}$ and b), we obtain that there is an accepting configuration set $H \subseteq ACC_{A,s}$, such that $\langle q_\nu, (u, [u_1; \ldots; u_l; u]) \rangle \vdash^+_{A,s} H$.

d) By (*) and c) we obtain that $\langle q_\mu, (u, [u_1; \ldots; u_l]) \rangle \vdash^+_{A,s} H$.

Thus, $u \in T_\mu$, which confirms property (5) of Definition 4.1 for node sets $T_{m_l}, \ldots, T_{m_{l+1}-1}$ with respect to A, s, $u_1, \ldots, u_l$, and $T_0, \ldots, T_{m_l-1}$. With this, we have finished the proof of this lemma. $\diamond$

In the following we show that the acceptance of A can be described in an alternative but equivalent way in terms of closed node sets.

Lemma 4.5 Let $0 \leq l \leq n$, $u, u_1, \ldots, u_l \in pos(s)$, and $m_l \leq \mu \leq m_{l+1} - 1$. The following statements are equivalent.

(a) There is a set of accepting configurations $H \subseteq ACC_{A,s}$, such that $\langle q_\mu, (u, [u_1; \ldots; u_l]) \rangle \vdash^*_{A,s} H$.

(b) For every $S_0, \ldots, S_{m_{l+1}-1} \subseteq pos(s)$, if the node sets $S_0, \ldots, S_{m_{l+1}-1}$ are strongly l-closed with respect to A, s, and $u_1, \ldots, u_l$, then $u \in S_\mu$.

Proof. (direction "(a) $\Rightarrow$ (b)":) Let $k \geq 1$ and suppose that $\langle q_\mu, (u, [u_1; \ldots; u_l]) \rangle \vdash^k_{A,s} H$ and that $S_0, \ldots, S_{m_{l+1}-1} \subseteq pos(s)$ are strongly l-closed with respect to $u_1, \ldots, u_l$. We prove by induction on k.

(i) Let $k = 1$. Then obviously, H is singleton and $\langle q_\mu, (u, [u_1; \ldots; u_l]) \rangle \vdash_{A,s} \langle q_{yes}, (u, [u_1; \ldots; u_l]) \rangle = H$. By property (1) of Definition 4.1 we obtain that $u \in S_\mu$.

(ii) Let $k > 1$. Then we consider the following cases.

case 1: $\langle q_\mu, (u, [u_1; \ldots; u_l]) \rangle \vdash_{A,s} \langle q_\nu, (u', [u_1; \ldots; u_l]) \rangle \vdash^{k-1}_{A,s} H$, where $m_l \leq \nu \leq m_{l+1} - 1$. Then, by the induction hypothesis $u' \in S_\nu$. Hence, by property (2) of Definition 4.1 we obtain that $u \in S_\mu$.

case 2: $\langle q_\mu, (u, [u_1; \ldots; u_l]) \rangle \vdash_{A,s} \{\langle q_{\nu_1}, (u, [u_1; \ldots; u_l]) \rangle, \langle q_{\nu_2}, (u, [u_1; \ldots; u_l]) \rangle\}$ $\vdash^{k-1}_{A,s} H$, where $m_l \leq \nu_1, \nu_2 \leq m_{l+1} - 1$. Then, it is obvious that there are accepting configuration sets $H_1, H_2 \subseteq ACC_{A,s}$ and numbers $k_1, k_2 < k$ such that $\langle q_{\nu_1}, (u, [u_1; \ldots; u_l]) \rangle \vdash^{k_1}_{A,s} H_1$ and $\langle q_{\nu_2}, (u, [u_1; \ldots; u_l]) \rangle \vdash^{k_2}_{A,s} H_2$. By the induction hypothesis $u \in S_{\nu_1}$ and $u \in S_{\nu_2}$. Hence, by property (3) of Definition 4.1 we obtain that $u \in S_\mu$.

case 3: $l \geq 1$ and $\langle q_\mu, (u, [u_1; \ldots; u_l]) \rangle \vdash_{A,s} \langle q_\nu, (u, [u_1; \ldots; u_{l-1}]) \rangle \vdash^{k-1}_{A,s} H$, where $m_{l-1} \leq \nu \leq m_l - 1$. Then, by the induction hypothesis $u \in S_\nu$. Hence, by property (4) of Definition 4.1, we obtain that $u \in S_\mu$.

case 4: $l < n$ and $\langle q_\mu, (u, [u_1; \ldots; u_l]) \rangle \vdash_{A,s} \langle q_\nu, (u, [u_1; \ldots; u_l; u]) \rangle \vdash^{k-1}_{A,s} H$, where $m_{l+1} \leq \nu \leq m_{l+2} - 1$. Let $S_{m_{l+1}}, \ldots, S_{m_{l+2}-1} \subseteq pos(s)$ be arbitrary $l+1$-closed node sets with respect to A, s, $u_1, \ldots, u_l, u$, and $S_0, \ldots, S_{m_{l+1}-1}$. Then, by the induction hypothesis $u \in S_\nu$. Hence, by property (5) of Definition 4.1 we obtain that $u \in S_\mu$.

(direction "(b) $\Rightarrow$ (a)":)

Let $u_{l+1}, \ldots, u_n \in pos(s)$ be arbitrary dummy nodes, and $T_0, \ldots, T_m \subseteq pos(s)$ the node sets defined in the same way as in Lemma 4.4. By that lemma, $T_0, \ldots, T_m$ are strongly n-closed with respect to A, s, and $u_1, \ldots, u_n$. From this fact and Definition 4.2 we obtain the following statement.

Statement: The node sets $T_0, \ldots, T_{m_{l+1}-1}$ are strongly l-closed with respect to A, s, and $u_1, \ldots, u_l$.

Now, assume that (b) holds. By (b) and our Statement we get that $u \in T_\mu$. It follows that there is an accepting configuration set $H \subseteq ACC_{A,s}$, such that $\langle q_\mu, (u, [u_1; \ldots; u_l]) \rangle \vdash^+_{A,s} H$, and with this, we have finished the proof. ◇

Corollary 4.6 $s \in L(A)$ if and only if for all 0-closed node sets $S_0, \ldots, S_{m_1-1}$, we have that $\varepsilon \in S_0$.

Proof.

$$\begin{aligned}
& s \in L(A) \\
\overset{\text{(Definition 3.9)}}{\Longleftrightarrow}\quad & \exists H \subseteq ACC_{A,s} \text{ such that } \langle q_0, (\varepsilon, [\,])\rangle \vdash_{A,s}^{*} H \\
\overset{\text{(Lemma 4.5)}}{\Longleftrightarrow}\quad & \text{for all strongly 0-closed node sets } S_0, \ldots, S_{m_1-1}, \\
& \text{we have } \varepsilon \in S_0 \\
\overset{\text{(Observation 4.3)}}{\Longleftrightarrow}\quad & \text{for all 0-closed node sets } S_0, \ldots, S_{m_1-1}, \\
& \text{we have } \varepsilon \in S_0. \qquad \diamond
\end{aligned}$$

Now we are ready to prove the main result of this section.

Theorem 4.7 For every $n \geq 0$, n-patwa recognize exactly the regular tree languages. Formally, $REG = n\text{-}PATWA$.

Proof. Clearly, already 0-patwa are capable to simulate classical top-down tree automata, hence each regular tree language is recognizable by an n-patwa for $n \geq 0$, i.e., $REG \subseteq n\text{-}PATWA$ for $n \geq 0$. For the converse, it suffices to prove that $L(A)$ is a regular tree language (since A is picked as an arbitrary n-patwa).

Now we construct an MSO-formula defining $L(A)$. The thorough reader will find this formula almost literally the same as the one in the proof of Theorem 4.7 of [21].

Let $b \in \{0,1\}^{\leq n}$ be a bitvector of length l. We define a predicate $pebbles_b(x, x_1, \ldots, x_l)$ with free variables $x, x_1, \ldots, x_l$ which is true in a structure (s, Π_1, Π_2), if the presence of pebbles at node $\Pi_1(x)$ agrees with b, assuming that l pebbles are on the input tree and the positions of pebbles $1, \ldots, l$ are $\Pi_1(x_1), \ldots, \Pi_1(x_l)$, respectively. The predicate $pebbles_b(x)$ is defined by induction on l.

(i) If $l = 0$, then $pebbles_b(x) = pebbles_\varepsilon(x) = true$.

(ii) If $l > 0$, then

$$pebbles_b(x, x_1, \ldots, x_l) = \begin{cases} pebbles_{b'}(x, x_1, \ldots, x_{l-1}) \wedge (x_l = x) & \text{if } b = b'1 \\ pebbles_{b'}(x, x_1, \ldots, x_{l-1}) \wedge \neg(x_l = x) & \text{if } b = b'0 \end{cases}$$

For every $\sigma \in \Sigma$, $b \in \{0,1\}^{\leq n}$, where $|b| = l$, and $0 \leq j \leq maxrank(\Sigma)$ let

$$\theta_{\sigma,b,j}(x) = lab_\sigma(x) \wedge pebbles_b(x, x_1, \ldots, x_l) \wedge chno_j(x)$$

be the formula with free first-order variables $x, x_1, \ldots, x_l$, which is true in a structure (s, Π_1, Π_2) iff the test result of the pebble configuration $(\Pi_1(x), [\Pi_1(x_1); \ldots; \Pi_1(x_l)])$ is (σ, b, j), see Definition 3.5.

For each $0 \leq l \leq n$ and $m_l \leq \mu \leq m_{l+1} - 1$, we give the formula $\phi_\mu^{(l)}$ as follows.

$$\phi_\mu^{(l)} = \begin{cases} \forall X_0 \ldots \forall X_{m_1-1} \Big(0\text{-}closed \rightarrow root \in X_\mu\Big) & \text{if } l = 0 \\ \forall X_{m_l} \ldots \forall X_{m_{l+1}-1} \Big(l\text{-}closed \rightarrow x_l \in X_\mu\Big) & \text{if } l > 0, \end{cases}$$

where

$$l\text{-}closed = \bigwedge_{r \in R_l} \psi_r,$$

and ψ_r's are defined as follows.

(1) If r is an accepting rule of the form $\langle q_\mu, \sigma, b, j\rangle \to \langle q_{yes}, \mathit{stay}\rangle$, where $q_\mu \in Q_l$, $0 \le l \le n$, and $\sigma \in \Sigma$, $b \in \{0,1\}^l$, $0 \le j \le \mathit{maxrank}(\Sigma)$, then

$$\psi_r = \forall x_{l+1}\Big(\theta_{\sigma,b,j}(x_{l+1}) \to x_{l+1} \in X_\mu\Big).$$

(2) If r is of the form $\langle q_\mu, \sigma, b, j\rangle \to \langle q_\nu, \mathit{stay}\rangle$, where $q_\mu, q_\nu \in Q_l$, $0 \le l \le n$, $\sigma \in \Sigma$, $b \in \{0,1\}^l$, and $0 \le j \le \mathit{maxrank}(\Sigma)$, then

$$\psi_r = \forall x_{l+1}\Big((\theta_{\sigma,b,j}(x_{l+1}) \wedge x_{l+1} \in X_\nu) \to x_{l+1} \in X_\mu\Big).$$

(3) If r is of the form $\langle q_\mu, \sigma, b, j\rangle \to \langle q_\nu, \mathit{up}\rangle$, where $q_\mu, q_\nu \in Q_l$, $0 \le l \le n$, $\sigma \in \Sigma$, $b \in \{0,1\}^l$, and $0 \le j \le \mathit{maxrank}(\Sigma)$, then

$$\psi_r = \forall x_{l+1}\forall y\Big((\theta_{\sigma,b,j}(x_{l+1}) \wedge \mathit{child}(x_{l+1}, y) \wedge y \in X_\nu) \to x_{l+1} \in X_\mu\Big).$$

(4) If r is of the form $\langle q_\mu, \sigma, b, j\rangle \to \langle q_\nu, \mathit{down}_i\rangle$, where $q_\mu, q_\nu \in Q_l$, $0 \le l \le n$, $\sigma \in \Sigma$, $b \in \{0,1\}^l$, and $0 \le j \le \mathit{maxrank}(\Sigma)$, then

$$\psi_r = \forall x_{l+1}\forall y\Big((\theta_{\sigma,b,j}(x_{l+1}) \wedge \mathit{child}_i(y, x_{l+1}) \wedge y \in X_\nu) \to x_{l+1} \in X_\mu\Big).$$

(5) If r is an alternating rule of the form $\langle q_\mu, \sigma, b, j\rangle \to \{\langle q_{\nu_1}, \mathit{stay}\rangle, \langle q_{\nu_2}, \mathit{stay}\rangle\}$, where $q_\mu, q_{\nu_1}, q_{\nu_2} \in Q_l$, $0 \le l \le n$, $\sigma \in \Sigma$, $b \in \{0,1\}^l$, and $0 \le j \le \mathit{maxrank}(\Sigma)$, then

$$\psi_r = \forall x_{l+1}\Big((\theta_{\sigma,b,j}(x_{l+1}) \wedge x_{l+1} \in X_{\nu_1} \wedge x_{l+1} \in X_{\nu_2}) \to x_{l+1} \in X_\mu\Big).$$

Moreover:

(6) If r is of the form $\langle q_\mu, \sigma, b, j\rangle \to \langle q_\nu, \mathit{lift}\rangle$, where $q_\mu \in Q_l$, $1 \le l \le n$, $\sigma \in \Sigma$, $b \in \{0,1\}^l$, $0 \le j \le \mathit{maxrank}(\Sigma)$, and $q_\nu \in Q_{l-1}$, then

$$\psi_r = \forall x_{l+1}\Big((\theta_{\sigma,b,j}(x_{l+1}) \wedge x_{l+1} \in X_\nu) \to x_{l+1} \in X_\mu\Big).$$

(7) If r is of the form $\langle q_\mu, \sigma, b, j\rangle \to \langle q_\nu, \mathit{drop}\rangle$, where $q_\mu \in Q_l$, $0 \le l \le n-1$, $\sigma \in \Sigma$, $b \in \{0,1\}^l$, $0 \le j \le \mathit{maxrank}(\Sigma)$, and $q_\nu \in Q_{l+1}$, then

$$\psi_r = \forall x_{l+1}\Big((\theta_{\sigma,b,j}(x_{l+1}) \wedge \phi_\nu^{(l+1)}) \to x_{l+1} \in X_\mu\Big).$$

We make the following observations concerning the formula $\phi_\mu^{(l)}$.

a) $\phi_\mu^{(l)}$ has free node-set variables $X_0, \ldots, X_{m_l-1}$, and free node variables $x_1, \ldots, x_l$.

(In particular, $\phi_\mu^{(0)}$ is a closed formula.)

b) The subformula *l-closed* of $\phi_\mu^{(l)}$ has free node-set variables $X_{m_l}, \ldots, X_{m_{l+1}-1}$, in addition to the free variables above, and *l-closed* is true in a structure (s, Π_1, Π_2) if and only if $\Pi_2(X_{m_l}), \ldots, \Pi_2(X_{m_{l+1}-1})$ are l-closed with respect to A, s, $\Pi_1(x_1), \ldots, \Pi_1(x_l)$ and $\Pi_2(X_0), \ldots, \Pi_2(X_{m_l-1})$.

Note that the conjunction of formulas of type (1)-(7) expresses Definition 4.1 for node sets $\Pi_2(X_{m_l}), \ldots, \Pi_2(X_{m_{l+1}-1})$ (with respect to A, s, $\Pi_1(x_1), \ldots, \Pi_1(x_l)$ and $\Pi_2(X_0), \ldots, \Pi_2(X_{m_l-1})$).

c) Hence, $\phi_\mu^{(l)}$ is true in a structure (s, Π_1, Π_2) if for all node-sets $S_{m_l}, \ldots, S_{m_{l+1}-1} \subseteq pos(s)$, l-closed with respect to A, s, $\Pi_1(x_1), \ldots, \Pi_1(x_l)$, and $\Pi_2(X_0), \ldots, \Pi_2(X_{m_l-1})$, we have that

- $root \in S_\mu$, if $l = 0$, or
- $\Pi_1(x_l) \in S_\mu$, if $l > 0$.

Thus, by Corollary 4.6, we obtain that $s \in L(A)$ if and only if $s \models \phi_0^{(0)}$. Hence, $L(A) = L(\phi_0^{(0)})$ and this concludes that the tree language recognized by A is MSO-definable, and thus it is regular. ◇

5 Inclusion results for patwa

In this section we investigate the recognizing power of deterministic and non-looping patwa with and without pebbles. First we collect the preliminary results, which are necessary for this section.

Theorem 1 of [4] says that deterministic tree-walking automata are less powerful than their nondeterministic counterparts. Formally, $dTWA \subset TWA$. We note that the separating tree language treated by [4] (which cannot be recognized by a deterministic twa) can be recognized already by a nondeterministic and non-looping twa. Thus, $dTWA_{nl} \subset TWA_{nl}$, and moreover, by Proposition 1 of [22] (saying that $dTWA = dTWA_{nl}$), we obtain the following "non-looping version" of the above proper inclusion result.

Proposition 5.1 *$dTWA \subset TWA_{nl}$.* ◇

Theorem 1. of [22] states that deterministic twa are closed under complementation.

Proposition 5.2 *$dTWA = co\text{-}dTWA$.* ◇

One of the main results of [5] is Theorem 1.1 saying that ptwa do not recognize all the regular tree languages, formally, $PTWA \subset REG$. Using the obvious fact that $PTWA_{nl} \subseteq PTWA$, we obtain the following proposition.

Proposition 5.3 *$PTWA_{nl} \subset REG$.* ◇

Moreover Theorem 1.2 of [5] says, that the expressive power of n-patwa is strictly less than the expressive power of $(n+1)$-patwa for each $n \geq 0$, formally, $n\text{-}PTWA \subset (n+1)\text{-}PTWA$. We note that Theorem 1.2 of [5] refines Theorem 2 of [3], which says that $TWA \subset REG$. However, we wish to obtain the "non-looping version" of the proper inclusion $n\text{-}PTWA \subset (n+1)\text{-}PTWA$. For this we make the following observations.

(1) In the preceding paragraph of Theorem 3 of [22] it was shown that for each n-ptwa A with weak pebble handling we can construct a non-looping n-ptwa A' with weak pebble handling, such that $L(A) = L(A')$.

(2) It was shown in Lemma 5.1 of [5] that for each n-ptwa A we can construct an n-ptwa A' with weak pebble handling, such that $L(A) = L(A')$.

(3) By Theorem 1.2 of [5], $n\text{-}PTWA \subset (n+1)\text{-}PTWA$.

We note that the "weak pebble handling" property for ptwa is discussed in the Introduction. By (1)-(3) we conclude the following proposition.

Proposition 5.4 For each $n \geq 0$, $n\text{-}PTWA_{nl} \subset (n+1)\text{-}PTWA_{nl}$. $\diamond$

Now we prove that the complements of the tree languages of $n\text{-}dPATWA_{nl}$ form exactly the tree language class $n\text{-}PTWA_{nl}$.

Lemma 5.5 For each $n \geq 0$, $co\text{-}n\text{-}dPATWA_{nl} = n\text{-}PTWA_{nl}$.

Proof. $\underline{co\text{-}n\text{-}dPATWA_{nl} \subseteq n\text{-}PTWA_{nl}}$: Let $A = (Q, \Sigma, q_0, q_{yes}, R)$ be a deterministic and non-looping n-patwa. We construct the (nondeterministic, non-looping) n-ptwa $A' = (Q, \Sigma, q_0, q_{yes}, R')$ where R' is the smallest set of rules satisfying the following conditions.

- For each $q \in Q$, $\sigma \in \Sigma$, $b \in \{0,1\}^{\leq n}$, and $j \in \{0, \ldots, maxrank(\Sigma)\}$, if there is no rule in R with left-hand side $\langle q, \sigma, b, j\rangle$, then the accepting rule $\langle q, \sigma, b, j\rangle \rightarrow \langle q_{yes}, stay\rangle$ is in R'.
- For each pebble tree-walking rule $\langle q, \sigma, b, j\rangle \rightarrow \langle p, \varphi\rangle$ of R it is also in R'.
- For each alternating rule $\langle q, \sigma, b, j\rangle \rightarrow \{\langle p_1, stay\rangle, \langle p_2, stay\rangle\}$, the pebble tree-walking rules $\langle q, \sigma, b, j\rangle \rightarrow \langle p_1, stay\rangle$ and $\langle q, \sigma, b, j\rangle \rightarrow \langle p_2, stay\rangle$ are in R'.

Since M is non-looping, it is obvious that also M' is non-looping. The proof of $L(A') = \overline{L(A)}$ is straightforward, hence we omit it.

$\underline{n\text{-}PTWA_{nl} \subseteq co\text{-}n\text{-}dPATWA_{nl}}$: Let $A = (Q, \Sigma, q_0, q_{yes}, R)$ be a non-looping (nondeterministic) ptwa. We construct the deterministic patwa with general alternating rules $A' = (Q', \Sigma, q_0, q'_{yes}, R')$ (by Lemma 3.10 we are allowed to use general alternating rules) as follows.

- $Q' = Q \cup \{q_{yes}\}$, and
- R' is the smallest set of rules satisfying the following conditions.
  - For each $q \in Q$, $\sigma \in \Sigma$, $b \in \{0,1\}^{\leq n}$, and $j \in \{0, \ldots, maxrank(\Sigma)\}$, if there is no rule in R with left-hand side $\langle q, \sigma, b, j\rangle$, then the accepting rule $\langle q, \sigma, b, j\rangle \rightarrow \langle q'_{yes}, stay\rangle$ is in R'.

- For each $q \in Q$, $\sigma \in \Sigma$, $b \in \{0,1\}^{\leq n}$, and $j \in \{0, \ldots, \mathit{maxrank}(\Sigma)\}$, if $\{\langle q_1, \varphi_1\rangle, \ldots, \langle q_m, \varphi_m\rangle\}$ is the set of state-instruction pairs that are the right-hand sides of rules with left-hand side $\langle q, \sigma, b, j\rangle$, then the rule $\langle q, \sigma, b, j\rangle \to \{\langle q_1, \varphi_1\rangle, \ldots, \langle q_m, \varphi_m\rangle\}$ is in R'.

Again, it is obvious that A' is deterministic, non-looping, and we leave the proof $L(M) = L(M')$ to the reader. $\diamond$

Now we prove the following proper inclusion result.

Theorem 5.6 *$dTWA \subset dATWA_{nl}$.*

Proof. We prove by contradiction. Let us assume that $dTWA = dATWA_{nl}$. Then we make the following observations.

a) Obviously, $co\text{-}dTWA = co\text{-}dATWA_{nl}$.

b) By Proposition 5.2, $dTWA = co\text{-}dATWA_{nl}$.

c) By Lemma 5.5, $dTWA = TWA_{nl}$, which contradicts Proposition 5.1.

With this, we have proved this theorem. $\diamond$

Next we prove that with the deterministic and non-looping n-patwa are strictly weaker than deterministic and non-looping $(n+1)$-patwa are.

Theorem 5.7 For each $n \geq 0$, $n\text{-}dPATWA_{nl} \subset (n+1)\text{-}dPATWA_{nl}$.

Proof. The inclusion $n\text{-}dPATWA_{nl} \subseteq (n+1)\text{-}dPATWA_{nl}$ is obvious. We prove the proper inclusion by contradiction. Let us assume that $n\text{-}dPATWA_{nl} = (n+1)\text{-}dPATWA_{nl}$. Then, applying operation '*co*' to both sides of the equation we get that $co\text{-}n\text{-}dPATWA_{nl} = co\text{-}(n+1)\text{-}dPATWA_{nl}$. By Lemma 5.5 we obtain that $n\text{-}PTWA_{nl} = (n+1)\text{-}PTWA_{nl}$, which contradicts Proposition 5.4. $\diamond$

Finally, we prove, that deterministic and non-looping patwa do not recognize all the regular tree languages.

Theorem 5.8 *$dPATWA_{nl} \subset REG$.*

Proof. $dPATWA_{nl} \subseteq REG$ comes from Theorem 4.7 The proper inclusion is proved by contradiction. Let us assume that $dPATWA_{nl} = REG$. Then we make the following observations.

a) Applying the operation '*co*' to both sides, we obtain that $co\text{-}dPATWA_{nl} = co\text{-}REG$.

b) By Proposition 2.1 we obtain that $co\text{-}dPATWA_{nl} = REG$.

c) By Lemma 5.5 we get that $PTWA_{nl} = REG$, which contradicts Proposition 5.3. $\diamond$

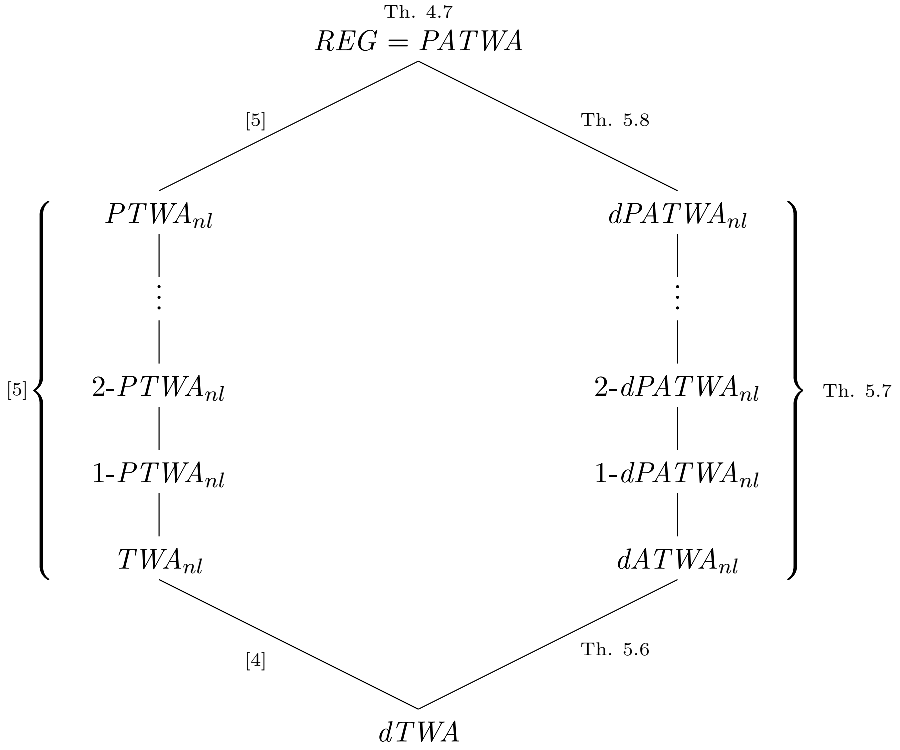

Figure 1: Inclusion diagram of some subclasses of *PATWA*. The continuous lines represent proper inclusions.

6 Conclusions

In this paper we gave a formal definition for pebble tree-walking automata extended with alternation [23] and have answered the open problem raised at page 18 in [10], which asked, whether the patwa recognize the class of regular tree languages. Our answer is yes, i.e., $PATWA = REG$.

In the remainder of this paper we have investigated the recognizing power of some subclasses of *PATWA*. We have come to the conclusion that

$$dTWA \subset dATWA_{nl} \subset 1\text{-}dPATWA_{nl} \subset \ldots \subset dPATWA_{nl} \subset REG.$$

However, it is still an open problem, whether $n\text{-}dPATWA \subset (n+1)\text{-}dPATWA$.

The most important known and new results are summarized in Figure 1.

We can find the relation between patwa and pebble tree transducers. It is trivial that the domain of each n-pebble tree transformation of [11] can be recognized by an n-patwa with *weak pebble handling*, i.e., pebbles can be lifted only from a node pointed at by the pointer. We can extend the pebble tree transducers of [11], such that the pebble in the input tree with the highest number can be lifted even from a node not pointed at by the pointer. This is the *strong pebble handling* (see

[10, 22, 5]), which is also used throughout the present paper. Then we can easily see that the domains of n-pebble tree transformations are exactly the tree languages recognized by n-patwa, i.e., the regular tree languages. Using the results of this paper, we obtain that

- $dom(n\text{-}PTT) = REG$, where $n \geq 0$, and
- $dTWA \subset dom(0\text{-}dPTT_{nl}) \subset dom(1\text{-}dPTT_{nl}) \subset \ldots \subset dom(dPTT_{nl}) \subset REG$,

assuming that $n\text{-}PTT$ is the class of n-pebble tree transformations, $PTT = \bigcup_{n\geq 0} n\text{-}PTT$ (the prefix '*d*', and the subscription '*nl*' denote the deterministic and non-looping subclasses), and for a tree transformation τ, the notation $dom(\tau)$ means the domain tree language of τ.

Another application of the inclusion result $dATWA_{nl} \subset 1\text{-}dPATWA_{nl} \subset REG$ is, that deterministic and non-looping atwa (0-patwa) are exactly the *tree-walking automata in universal acceptance mode* of [13], where it was proved that these automata recognize exactly the domains of partial attributed tree transformations [12]. It is straightforward that the non-looping version for this result also holds, i.e., non-looping tree-walking automata in universal acceptance mode recognize exactly the tree languages recognized by non-looping deterministic *atwa*, that are the domains of non-looping partial attributed tree transformations. Hence, we obtain that $dTWA \subset dom(ATT_{nl}) \subset 1\text{-}dPATWA_{nl} \subset REG$, where ATT_{nl} is the class of non-looping partial attributed tree transformations.

Acknowledgments

I thank Zoltán Fülöp for giving me a lot of useful advice. Moreover, I am grateful to the anonymous referee for his/her valuable ideas and suggestions which have rised the quality of this paper substantially.

References

[1] A. V. Aho and J. D. Ullman. Translations on a context–free grammar. *Inform. Control*, 19:439–475, 1971.

[2] R. Bloem and J. Engelfriet. Characterization of properties and relations defined in monadic second order logic on the nodes of trees. Technical Report Technical Report 97-03, Leiden University, August 1997.

[3] M. Bojańczyk and T. Colcombet. Tree-walking automata do not recognize all regular languages. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 234–243, New York, NY, USA, 2005. ACM Press.

[4] M. Bojańczyk and T. Colcombet. Tree-walking automata cannot be determinized. *Theoretical Computer Science*, 350:164–173, 2006.

[5] M Bojańczyk, M. Samuelides, T. Schwentick, and L. Segoufin. Expressive power of pebble automata. In *ICALP'06: Proceedings of 33rd International Colloquium on Automata, Languages and Programming*, pages 157–168. Springer Berlin / Heidelberg, 2006.

[6] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[7] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[8] J. Doner. Tree acceptors and some of their applications. *J. Comput. System Sci.*, 4:406–451, 1970.

[9] J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 72–83, London, UK, 1999. Springer-Verlag.

[10] J. Engelfriet and Hendrik Jan Hoogeboom. Automata with nested pebbles capture first-order logic with transitive closure. Technical Report 05-02, Leiden University, The Netherlands, April 2005.

[11] J. Engelfriet and S. Maneth. A Comparison of Pebble Tree Transducers with Macro Tree Transducers. *Acta Informatica*, 39:613–698, 2003.

[12] Z. Fülöp. On attributed tree transducers. *Acta Cybernet.*, 5:261–279, 1981.

[13] Z. Fülöp and S. Maneth. Domains of partial attributed transducers. *Inform. Process. Letters*, 73:175–180, 2000.

[14] Z. Fülöp and L. Muzamel. Decomposition Results for Pebble Macro Tree Transducers. Technical Report TUD-FI05-13, Technical University of Dresden, 2005.

[15] Z. Fülöp and L. Muzamel. Circularity and Decomposition Results for Pebble Macro Tree Transducers. *submitted*, 2006.

[16] Z. Fülöp and H. Vogler. *Syntax-Directed Semantics — Formal Models Based on Tree Transducers*. Monographs in Theoretical Computer Science, An EATCS Series. Springer-Verlag, 1998.

[17] F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[18] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 1–68. Springer-Verlag, 1997.

[19] D. E. Knuth. Semantics of context-free languages: Correction. *Math. Systems Theory*, 5(1):95–96, 1971. Errata of [20].

[20] D.E. Knuth. Semantics of context–free languages. *Math. Systems Theory*, 2:127–145, 1968.

[21] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. *J. of Comp. Syst. Sci.*, 66:66–97, 2003.

[22] A. Muscholl, M. Samualides, and L. Segoufin. Complementing deterministic tree-walking automata. *Information Processing Letters*, 99:33–39, 2006.

[23] G. Slutzki. Alternating tree automata. *Theoretical Computer Science*, 41:305–318, 1985.

[24] J. W. Thatcher and J.B. Wright. Generalized finite automata theory with application to a decision problem of second-order logic. *Math. Systems Theory*, 2(1):57–81, 1968.