HAL
open science

Expressive power of pebbles Automata

Mikolaj Bojanczyk, Mathias Samuelides, Thomas Schwentick, Luc Segoufin

To cite this version:

Mikolaj Bojanczyk, Mathias Samuelides, Thomas Schwentick, Luc Segoufin. Expressive power of pebbles Automata. Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, 2006, Venice, Italy. pp.157-168. hal-00158631

HAL Id: hal-00158631

https://hal.science/hal-00158631v1

Submitted on 29 Jun 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Expressive power of pebble automata ⋆

Mikołaj Bojańczyk[1,2], Mathias Samuelides[2], Thomas Schwentick[3], Luc Segoufin[4]

[1] Warsaw University
[2] LIAFA, Paris 7
[3] Universität Dortmund
[4] INRIA, Paris 11

Abstract. Two variants of pebble tree-walking automata on trees are considered that were introduced in the literature. It is shown that for each number of pebbles, the two models have the same expressive power both in the deterministic case and in the nondeterministic case. Furthermore, nondeterministic (resp. deterministic) tree-walking automata with $n+1$ pebbles can recognize more languages than those with n pebbles. Moreover, there is a regular tree language that is not recognized by any tree-walking automaton with pebbles. As a consequence, FO+posTC is strictly included in MSO over trees.

1 Introduction

In this paper we study pebble automata on binary trees. A pebble automaton is a sort of sequential automaton which moves from node to node in a tree, along its edges. Besides a finite set of states it has a finite set $\{1, \ldots, n\}$ of pebbles which it can drop at and lift from nodes. There is a restriction though: pebble i can only be dropped at the current node if pebbles $i+1, \ldots, n$ are already on the tree. Likewise, if pebbles $i, \ldots, n$ are on the tree only pebble i can be lifted. Pebble automata were introduced in [4] as a model with intermediate expressive power between tree-walking automata [1, 7] and parallel bottom-up or top-down automata. They are closely related to some aspects of XML languages. Furthermore, they are a building block of *pebble transducers* which were used to capture XML transformations (cf. [8, 6]).

Besides the number of pebbles, there are other parameters of pebble automata that can be varied. For example, they may be deterministic or nondeterministic, and they may have different policies of lifting a pebble: in the original model [4], a pebble can be lifted only if it is at the current node (*head position*), in the strong model, which was used to obtain a logical characterization in [5], it can be lifted everywhere. Not much is known about the relationships between the classes induced by the different models. Until recently it was even conceivable that deterministic tree-walking automata (sequential automata *without* pebbles) could recognize all regular languages. In [2, 3] this has been refuted and it has been shown that nondeterministic tree-walking automata do not recognize all regular tree languages but are strictly more expressive than deterministic tree-walking automata.

The current paper sheds some more light on the relationship between the pebble automata classes. In a nutshell, (a) whether pebbles are strong or not does not change the expressive power but (b) increasing the number of pebbles or moving from the deterministic to the nondeterministic model increases the expressive power.

We next give an overview of the results of this paper. We write PA for the class of tree languages recognized by nondeterministic pebble automata. We add a subscript n for the restriction to n-pebble automata, 'D' to indicate deterministic automata and 's' for the strong model, e.g., sDPA_n is the class of tree languages recognized by deterministic strong n-pebble automata. REG denotes the class of regular tree languages.

⋆ Work supported by the French-German cooperation programme PROCOPE, KBN Grant 4 T11C 042 25, and the EU-TMR network GAMES.

Expressive power of pebble automata. The main result of this paper is that pebble automata do not recognize all regular tree languages.

Theorem 1.1. $\mathrm{PA} \subsetneq \mathrm{REG}$.

This result is refined by showing that the hierarchy for pebble automata based on the number of pebbles is strict for both nondeterministic and deterministic pebble automata.

Theorem 1.2. *For each* $n \geq 0$, $\mathrm{PA}_n \subsetneq \mathrm{PA}_{n+1}$ *and* $\mathrm{DPA}_n \subsetneq \mathrm{DPA}_{n+1}$.

This settles open questions raised in [4, 5]. Furthermore, for each n, there is a language recognized by a nondeterministic tree-walking automaton but not by a deterministic n-pebble automaton. This improves the result in [2] that tree-walking automata (pebble automata with no pebbles) can not always be determinized.

Theorem 1.3. *For each* $n \geq 0$, $\mathrm{TWA} \not\subseteq \mathrm{DPA}_n$.

It is still an open problem to know whether DPA is strictly included in PA or not.

Strong pebble automata. In [5], *strong* pebble automata were introduced as a model which corresponds to natural logics on trees. It was stated as an open question whether this model is stronger than the original one. We were surprised that this is actually not the case.

Theorem 1.4. *For each* $n \geq 0$, $\mathrm{sPA}_n = \mathrm{PA}_n$ *and* $\mathrm{sDPA}_n = \mathrm{DPA}_n$.

This proof is effective, but the state space increases n-fold exponentially. In a recent paper [9], it was shown that DPA_n is closed under complement but the closure under complement of sDPA_n was left open. Nevertheless, it was shown that the complement of a language in sDPA_n is in sDPA_{3n}. From Theorem 1.4 we get the following stronger result:

Corollary 1.5. *For each* $n \geq 0$, sDPA_n *is closed under complement.*

Consequences for logics. In [5], the expressive power of strong pebble automata has been characterized in terms of logics. It was shown that FO+DTC=sDPA and FO+posTC=sPA. Here, FO+DTC is the extension of first-order logic with unary deterministic transitive closure operators and FO+posTC is the extension with positive unary transitive closure operators. By combining these results with ours and the fact that the regular tree languages are captured by monadic second-order logic (MSO), we immediately obtain the following result.

Corollary 1.6. *FO+posTC* $\subsetneq$ *MSO.*

Whether FO+TC $\subsetneq$ MSO and FO+DTC $\subsetneq$ FO+posTC remains open.

In Section 2 we give precise definitions of pebble automata and develop some related terminology. In Section 3 we prove some basic facts about the behavior of pebble automata on trees, in particular we show a kind of universality of n-pebble automata: for each n-pebble automaton $\mathcal{A}$, there is an n-pebble automaton which on a tree t computes, in some sense, the complete behavior of $\mathcal{A}$ on t, for all possible contexts in which t may occur. In Section 4 we use these techniques to prove our separation results. Finally, in Section 5, we prove that strong pebbles give no additional power, thereby completing the proof of Corollary 1.6. Because of space limitation some proofs are missing and are available in the full version of this paper.

Acknowledgment. We are deeply indebted to Joost Engelfriet for carefully reading a previous draft of this paper and, in particular, pointing out a significant shortcoming in one of the proofs.

2 Definitions

We consider finite, binary trees labeled by a given finite alphabet Σ. We insist that each non-leaf node has exactly two children. A set of trees over a given alphabet is called a **tree language**. Given a tree t and a node v of t, we denote by $t|_v$ the Σ-tree corresponding to the subtree of t rooted at v. Let $*$ be a new symbol not in Σ. A **context** is a tree over $\Sigma \cup (\Sigma \times \{*\})$, where the label with $*$ occurs only once and at a leaf. This unique leaf whose label contains $*$ is called the **port** of the context. Given a context C and a tree t such that the label of the root of t is the same as the Σ-part of the label of the port of C, we denote by $C[t]$ the tree which is constructed from C and t by replacing the $*$-leaf with t. The context $C_{t,v}$ is the context resulting from t by removing all proper descendants of v and adding $*$ to the label of v.

Pebble automata. Informally, a pebble automaton – just like a tree walking automaton – *walks* through its input tree from node to node along the edges. Additionally it has a fixed set of pebbles, numbered from 1 to n that it can place in the tree. At each time, pebbles $i, \ldots, n$ are placed on some nodes of the tree, for some i. In one step, the automaton can stay at the current node, move to its parent, to its left or to its right child, or it can lift pebble i or place pebble $i-1$ on the current node. Which of these transitions can be applied depends on the current state, the label and the type of the current node (root, left or right child — leaf or inner node), the set of pebbles at the current node and the number i.

We consider two kinds of pebble automata which differ in the way they can lift a pebble. In the standard model a pebble can be lifted only if it is on the current node. In the **strong** model this restriction does not apply.

Remark 2.1.
- In both models the placement of the pebbles follows a stack discipline: only the pebble with the number i can be lifted and only the pebble with number $i-1$ can be placed. The restriction is essential as otherwise we would obtain n-head automata.
- The reader might wonder why the pebbles on the tree are numbered from n to i and not from 1 to i. The reason is simple: If pebbles $i, \ldots, n$ are on the tree, we can view the computation until pebble i is lifted as the computation of an $(i-1)$-pebble automaton, i.e., an automaton with pebbles $1, \ldots, i-1$. This will be convenient in proofs by induction on the number of pebbles already dropped on the tree.

We turn to the formal definition of pebble automata. The set $\text{types} = \{r, 0, 1\} \times \{l, i\}$ describes the possible types of a node. Here, r stands for the root, 0 for a left child, 1 for a right child, l for a leaf and i for an internal node (not a leaf). We indicate the possible kinds of moves of a pebble automaton by elements of the set $\{\epsilon, \uparrow, \swarrow, \searrow, \text{lift}, \text{drop}\}$, where informally $\uparrow$ stands for 'move to parent', ϵ stands for 'stay', $\swarrow$ for 'move to left child' and $\searrow$ for 'move to right child'. Clearly, drop refers to dropping a pebble and lift to lifting a pebble. Finally, $2^{[n]}$ denotes the powerset of $\{1, \ldots, n\}$.

Definition 2.2. An **n-pebble automaton** is a tuple $\mathcal{A} = (Q, \Sigma, I, F, \delta)$, where Q is a finite set of *states*, $I, F \subseteq Q$ are respectively the sets of *initial* and *accepting* states, and δ is the *transition relation* of the form

$$\delta \subseteq (Q \times \text{types} \times \{0, \ldots, n\} \times 2^{[n]} \times \Sigma) \times (Q \times \{\epsilon, \uparrow, \swarrow, \searrow, \text{lift}, \text{drop}\}).$$

A tuple $(q, \beta, i, S, \sigma, q', m) \in \delta$ intuitively means that if $\mathcal{A}$ is in state q with pebbles $i, \ldots, n$ on the tree, the current node has the pebbles from S, has type β and is labeled by σ then $\mathcal{A}$ can enter state q' and do a transition according to m.

A **pebble set** of $\mathcal{A}$ is a set $P \subseteq \{1, \ldots, n\}$. For a tree t, a **P-pebble assignment** is a function f which maps each $j \in P$ to a node in t. A **P-pebbled tree** is a tree t with an

associated P**-pebble assignment**. A **pebbled tree** is a P-pebbled tree, for some P. We usually do not explicitly denote f. Analogous notions are defined for contexts.

For $0 \leq i \leq n$, an i**-configuration** c is a tuple (v, q, f), where v is a node, q a state and f a $\{i+1, \ldots, n\}$-pebble assignment. We call v the **current node**, q the **current state** and f the **current pebble assignment**. We also write $(v, q, v_{i+1}, \ldots, v_n)$ if $f(j) = v_j$, for each $j \geq i+1$.

We write $c \vdash_{\mathcal{A},t} c'$ to denote that the automaton can make a (single step) transition from configuration c to c'. We denote the transitive closure of $\vdash_{\mathcal{A},t}$ by $\vdash^{+}_{\mathcal{A},t}$.

The relation $\vdash_{\mathcal{A},t}$ is basically defined in the obvious way following the intuition described above. However, there is a restriction of the lift-operation. A lift-transition can only be applied to an i-configuration (v, q, f) if $f(i+1) = v$, i.e., if pebble $i+1$ is at the current node. In Section 5 we also consider **strong** pebble automata for which this restriction does not hold.

A **run** is a nonempty sequence $c_1, \ldots, c_l$ of configurations such that $c_j \vdash_{\mathcal{A},t} c_{j+1}$ holds for each j. It is **accepting** if it starts and ends in the root of the tree with no pebble on the tree, the first state in I and the last state in F. The automaton $\mathcal{A}$ **accepts** a tree if it has an accepting run on it. A set of Σ-trees L is **recognized** by an automaton that accepts exactly the trees in L. Finally, we say that a pebble automaton is **deterministic** if δ is a function from $Q \times \text{types} \times \{1, \ldots, n\} \times 2^{[n]} \times \Sigma$ to $Q \times \{\epsilon, \uparrow, \swarrow, \searrow, \text{lift}, \text{drop}\}$.

We use PA_n (sPA_n) to denote the class of tree languages recognized by some (strong) pebble automaton using n pebbles and DPA_n (sDPA_n) for the corresponding deterministic classes. We write PA for $\bigcup_{n>0} \text{PA}_n$ and so forth.

Note that a (strong or standard) pebble automaton without pebbles is just a tree walking automaton. Thus, we also write TWA and DTWA for PA_0 and DPA_0, respectively.

An i**-run** is a run from an i-configuration to an i-configuration in which pebble $i+1$ is never lifted. An i**-loop** is an i-run from a configuration (v, p, f) to a configuration (v, q, f). Therefore, an i-loop is determined by the source i-configuration (v, p, f) and the target state q.

An i**-move** is an i-run with only two i-configurations: the first and last one. It can be (a) a single transition, or (b) a *drop* i transition, followed by an $(i-1)$-loop followed by a *lift* i transition. If the automaton is strong it can also be (c) *drop* i, followed by a (non-loop) $(i-1)$-run, followed by *lift* i.

3 Behaviors and how to compute them

Let an n-pebble automaton $\mathcal{A}$ be fixed for the rest of the section. It is important in this section that we work with a *standard* pebble automaton and not with a *strong* one.

Let v be a node in a tree t and $c = (v, p, v_{i+1}, \ldots, v_n)$ an i-configuration. Intuitively, whether or not there is an i-loop that starts in c clearly only depends on $t|_v$ and $C_{t,v}$ together with the pebble placement. Nevertheless, the exact relationship is not obvious: e.g., the automaton might enter $t|_v$, drop pebble i, move to $C_{t,v}$, drop pebble $i-1$ and then enter $t|_v$ again. Thus, the behavior of $\mathcal{A}$ depends on $t|_v$ and $C_{t,v}$ in an interleaving manner.

In this section, we will formalize the intuitive notion of *behavior* of $\mathcal{A}$ through the notion of simulation. Intuitively, a tree s is simulated by a tree t if all loops in s also exist in t. The *behavior* of a tree is its simulation equivalence class (the set of trees that both simulate it, and are simulated by it). We show that, for each $\mathcal{A}$, (1) there are only finitely many different behaviors, (2) behaviors are compositional, and (3) the behavior of a tree can be computed by another pebble automaton with the same number of pebbles.

Two pebble assignments f and g are i**-compatible** if their domains partition $\{i+1, \ldots, n\}$. A pebbled tree t with assignment f is i**-compatible** with a pebbled context C with assignment g if f and g are i-compatible and the pebbles assigned to the root of t by f are exactly the pebbles assigned to the port of C by g.

Given a pebbled context C and an i-compatible pebbled tree t, let $\mathbf{loops}_i(C,t)$ denote the set of pairs (p,q) for which there is an i-loop ρ in $C[t]$ from $(v,p,f\cup g)$ to $(v,q,f\cup g)$, where v is the junction node between C and t. An i-loop is a **tree i-loop** if it involves no i-configurations outside t, it is a **context i-loop** if it involves no i-configurations outside C. By $\mathbf{tree\text{-}loops}_i(C,t)$ ($\mathbf{context\text{-}loops}_i(C,t)$) we denote the corresponding set where ρ is a tree (context) i-loop. Clearly, $\mathbf{tree\text{-}loops}_i(C,t)\cup\mathbf{context\text{-}loops}_i(C,t)\subseteq\mathbf{loops}_i(C,t)$.

We next formalize the intuition that a tree s has all behaviors that a tree t has.

Definition 3.1. Let t,s be P-pebbled trees, for some P. We say s is **i-simulated** by t if, for every i-compatible pebbled context C, $\mathbf{tree\text{-}loops}_i(C,s)\subseteq\mathbf{tree\text{-}loops}_i(C,t)$.

We define i-simulation of pebbled contexts analogously. If s is j-simulated by t, for every $j\in\{0,\ldots,i\}$ we say that s is **i^*-simulated** by t.

3.1 Finitely many behaviors

Two P-pebbled trees (resp. contexts) are said to be **i-equivalent** if they i-simulate each other; they are **i^*-equivalent** if they i^*-simulate each other. We will denote context equivalence classes by γ and tree equivalence classes by τ. We write $\tau_i(t)$ (resp. $\gamma_i(C)$) for the i^*-equivalence class of a pebbled tree t (resp. pebbled context C). We show in this subsection that there are only finitely many i-equivalence classes (and therefore finitely many i^*-equivalence classes.) The following technical lemma shows that the notion of i^*-simulation actually also covers context i-loops, not only tree i-loops.

Lemma 3.2. *Let $i\le n$. Let s,t be pebbled trees and C a pebbled context, such that s and t are i-compatible with C.*

1. $\mathbf{context\text{-}loops}_0(C,s)=\mathbf{context\text{-}loops}_0(C,t)$
2. *For $i>0$, if s is $(i-1)^*$-simulated by t, then* $\mathbf{context\text{-}loops}_i(C,s)\subseteq\mathbf{context\text{-}loops}_i(C,t)$.
3. *If s is i^*-simulated by t, then* $\mathbf{loops}_i(C,s)\subseteq\mathbf{loops}_i(C,t)$.

Proof. We first show that (1) and (2) implies (3). An arbitrary i-loop in $C[s]$ can be decomposed into a number of tree i-loops in $C[s]$ and a number of context i-loops in $C[s]$. The tree i-loops in $C[s]$ exist in $C[t]$ because s is i-simulated by t. The context i-loops in $C[s]$ exist in $C[t]$ because of (1) and (2). It remains to show (1) and (2). Item (1) is obvious, since a context 0-loop in $C[s]$ (or $C[t]$) only visits C. For item (2), let ρ be a context i-loop in $C[s]$ from p to q. We decompose the run ρ into a sequence $\rho_0,\pi_1,\rho_1,\ldots,\pi_m,\rho_m$, where in the ρ_k neither the head nor a pebble $\le i$ is outside C and each π_k is a tree j-loop in $C'[s]$, for some $j<i$, where C' is the context extending C with pebbles $\{j+1,\ldots,i\}$. Since s is j-simulated by t, $\mathbf{tree\text{-}loops}_j(C',s)\subseteq\mathbf{tree\text{-}loops}_j(C',t)$, thus there is a corresponding tree j-loop π'_k in $C'[t]$. Clearly, the runs ρ_k also exist in $C[t]$ (modulo the assignment of pebbles $>i$ in s and t). Thus, there is a context i-loop ρ' in $C[t]$ with the same initial and final states. □

We associate with every tree i^*-equivalence class τ a (pebbled) tree t_τ of this class and likewise we choose a (pebbled) context C_γ, for each γ. If γ is a $(i-1)^*$-equivalence class, then from the dual of Lemma 3.2(2) we can conclude that $\mathbf{tree\text{-}loops}_i(C,t)=\mathbf{tree\text{-}loops}_i(C_\gamma,t)$, for every context C of class γ.

Given a pebbled tree t, its **tree i-behavior** B^i_t, for $i>0$, is a function that maps $(i-1)^*$-equivalence class γ to the set of pairs $\mathbf{tree\text{-}loops}_i(C_\gamma,t)$. It is defined only for γ such that C_γ is i-compatible with t. For $i=0$, B^i_t is simply the set of tree 0-loops of t. The context i-behavior B^i_C is defined analogously.

There is a natural order on i-behaviors: $B^i_s\le B^i_t$ if $B^i_s(\gamma)\subseteq B^i_t(\gamma)$ holds for all γ. The following technical lemma shows that the i-behaviors completely determine the i-equivalence classes and their simulations:

Lemma 3.3. *Let s, t be P-pebbled trees, for some P. Then $B^i_s \leq B^i_t$ iff s is i-simulated by t.*

Proof. By definition of B^i_t and the remark following Lemma 3.2 we have $B^i_t(\gamma_{i-1}(C)) = \textbf{tree-loops}_i(C, t)$ for every pebbled context C that is i-compatible with t. Hence, s is i-simulated by t iff $B^i_s(\gamma_{i-1}(C)) \subseteq B^i_t(\gamma_{i-1}(C))$ for every such C iff $B^i_s \leq B^i_t$. □

Thus, $B^i_s = B^i_t$ iff s and t are i-equivalent and from now on we also refer to the i-equivalence class of a tree as its tree i-behavior. A simple inductive argument shows:

Lemma 3.4. *For each $i \leq n$, there are finitely many tree (resp. context) i-equivalence classes.*

Proof. The proof is by induction on i. For $i = 0$ it is clear. Let $i > 0$ and assume the lemma is proved for all $j < i$. By induction the number of $(i-1)^*$-equivalence classes is finite and therefore the number of i-behaviors is finite. By Lemma 3.3 this implies that the number of i-equivalence classes is finite. □

The above construction is nonelementary, and this cannot be improved. One can easily show that the number of behaviors is at least as big as the smallest depth of an accepted tree. Using a standard construction for first-order logic, one can construct an n-pebble automaton with $O(n)$ states that only accepts trees whose depth is a tower of n exponentials.

3.2 Behaviors are compositional

We show next that i-behaviors behave compositionally. For instance, the i-behavior of a tree depends only on the i-behaviors of its two subtrees and the label of the root.

Let R, P_0, P_1 be a partition of $\{i+1, \ldots, n\}$ and let a be a label. For trees t_0, t_1 pebbled with P_0, P_1, respectively, we write $\textbf{Compose}(a, R, t_0, t_1)$ for the pebbled tree consisting of an a-labeled and R-pebbled root which has t_0 and t_1 as left and right subtrees, respectively. Similarly, for a P_0-pebbled tree t and a P_1-pebbled context C, $\textbf{Compose}(C, a, R, t, *)$ is the context composed from C and t as illustrated in Fig. 1. Likewise, $\textbf{Compose}(C, a, R, *, t)$ is the context where the port is the left sibling of t.

Given ordered sets A, B, C, an operation $f : A \times B \to C$ is **monotone** if $a \leq a', b \leq b'$ implies $f(a, b) \leq f(a', b')$.

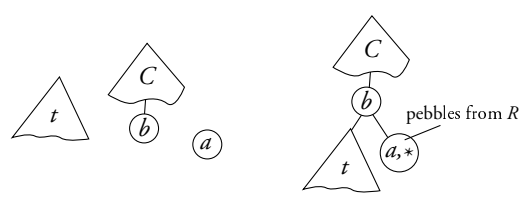


Fig. 1. The left-composed context $\textbf{Compose}(C, a, R, t, *)$.

Lemma 3.5. *Once the label a and pebble set R are fixed, the composition operations are monotone with respect to i^*-simulation.*

In particular, i^*-equivalence is a congruence for the composition operations. Thus, it makes sense to write $\textbf{Compose}(a, R, \tau_0, \tau_1)$ for the i^*-equivalence class of any tree with an a-labeled, R-pebbled root and subtrees of i^*-equivalence class τ_0 and τ_1. The proof of Lemma 3.5 is by induction on i and is straightforward by composing the subruns of the automaton in each of the subcomponents.

3.3 Behaviors can be calculated

In the following lemma we assume that pebbles $i+1, \ldots, n$ in a tree are suitably encoded by an (enlarged) alphabet. The proof is omitted in this abstract.

Lemma 3.6. *For every $i \leq n$ and tree i-behavior B^i, there is an i-pebble automaton $\mathcal{A}'$ that recognizes the pebbled trees t with $B^i_t \geq B^i$. Likewise for contexts. If $\mathcal{A}$ is deterministic, $\mathcal{A}'$ can be chosen deterministic, as well.*

3.4 Behavior foldings

In this section, we show one more closure property of pebble automata. For $i \geq 0$, the **i^*-behavior** of a tree t is defined as the sequence $B_t^0, \ldots, B_t^i$ (or, equivalently, the i^*-equivalence class of t; see the paragraph after Lemma 3.3). An i^*-behavior folding of a tree t is a tree that is obtained from t by replacing, for some nodes v of t, the subtree $t|_v$ with a single node labeled by the i^*-behavior of $t|_v$.

The techniques from Lemmas 3.6 can be generalized to i^*-behavior foldings:

Lemma 3.7. *For every $i \leq n$ and tree i-behavior B^i, there is an i-pebble automaton $\mathcal{B}$ that recognizes the i^*-behavior foldings of pebbled trees t with $B_t^i \geq B^i$. Likewise for contexts. If $\mathcal{A}$ is deterministic, $\mathcal{B}$ can be chosen deterministic, as well.*

Proof. Induction on i. We take the automaton $\mathcal{A}'$ from Lemma 3.6, which tests if a tree has behavior B^i, and simulate it over the i^*-behavior folding. Whenever the simulated automaton is in a leaf v of a behavior folding that has some i^*-behavior τ written in it, the simulating automaton $\mathcal{B}$ non-deterministically determines the possible tree j-loops (for $j \leq i$) that could be made in this node in the original tree. These loops depend on τ, and on the $(j-1)^*$-behavior γ of the rest of the tree (the $(j-1)^*$-behavior of the context $C_{t,v}$ along with the pebble assignment for pebbles $\{j+1, \ldots, i\}$). We note for reference in Lemma 4.6 the type of information used here: in the leaf v, only the part of the folded label τ with the j^*-behavior is used, while in the rest of the tree $C_{t,v}$, only the $(j-1)^*$-behaviors are read from the folded labels. We now mark node v using pebble j and compute the behavior γ using the remaining $j-1$ pebbles thanks to the induction assumption.

The deterministic case can be adapted as in Lemma 3.6. □

4 The pebble automata hierarchy

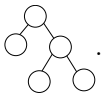
In this section we will prove Theorems 1.1, 1.2 and 1.3. In Subsection 4.1, we define the separating tree languages that we will use. In Subsection 4.2 we introduce *oracle automata*, a slight extension of tree-walking automata and show that the results (cf. Theorem 4.1 below) of [2] and [3] can be generalized to these models. Finally, in Subsection 4.3 we show the mentioned results.

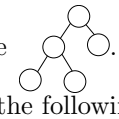
4.1 The separating languages

In this section, we will mostly deal with trees over the alphabet $\{\mathbf{a}, \mathbf{b}\}$. Moreover we require that only leaves can be labeled by **a**. We call these trees **quasi-blank trees**. An inner node of a quasi-blank tree is labeled by **b** and a leaf of a quasi-blank tree is labeled either by **a** or **b**.

For a quasi-blank tree t we define its **branching structure** $b(t)$. The branching structured results from t by first removing all nodes from t besides the **a**-labelled leaves and their ancestors. Then, all inner nodes with only one child are removed. Thus, $b(t)$ consists only of the **a**-leaves, and of deepest common ancestors of **a**-leaves. Note that the descendant-relation of the nodes of $b(t)$ is inherited from t.

By $\mathcal{L}_{\text{branch}}$ we denote the set of quasi-blank trees t such that all the paths from root to leaf of $b(t)$ have even length.

Let $\mathcal{L}_{3\text{l}}$ be the set of quasi-blank trees t such that $b(t)$ is  . Thus a quasi-blank tree in $\mathcal{L}_{3\text{l}}$ has exactly three **a**-leaves whose branching structure corresponds to the tree depicted

above. Likewise, $\mathcal{L}_{3\mathrm{r}}$ is the language of trees with branching structure  . Note that each quasi-blank tree with 3 **a**-leaves is either in $\mathcal{L}_{3\mathrm{l}}$ or in $\mathcal{L}_{3\mathrm{r}}$. We use the following result.

Theorem 4.1. *$\mathcal{L}_{3\mathrm{l}}$ and $\mathcal{L}_{3\mathrm{r}}$ are in* TWA *but not in* DTWA *[2]. $\mathcal{L}_{\mathrm{branch}}$ is in* REG *but not in* TWA *[3].*

Actually, in [3] a slightly stronger result was shown: for each TWA $\mathcal{A}$, there are trees $s' \in \mathcal{L}_{\mathrm{branch}}$ and $t' \notin \mathcal{L}_{\mathrm{branch}}$ such that each root-to-root loop of $\mathcal{A}$ in s' also exists in t'.

For the construction in this section we would need yet a stronger statement, namely that s' and t' have *the same* root-to-root loops. To this end, we define another tree language $\mathcal{L}_{\mathrm{even}}$ on top of $\mathcal{L}_{\mathrm{branch}}$, as follows. We recall that in a finite binary tree each node can be naturally addressed by a $\{0,1\}$-string describing the path from the root to the node where 0 corresponds to taking the left child of a node. In that spirit, a 0^*1-*node* is a right child of a node of the leftmost path. Let $\mathcal{L}_{\mathrm{even}}$ be the set of trees t for which $b(t)$ has an even number of 0^*1-nodes v whose subtree has all branches of even length.

We claim that $\mathcal{L}_{\mathrm{even}}$ has the desired property.

Proposition 4.2. *For every TWA $\mathcal{A}$, there are trees $s \in \mathcal{L}_{\mathrm{even}}$ and $t \notin \mathcal{L}_{\mathrm{even}}$ which have the same root-to-root loops of $\mathcal{A}$.*

Proof. Let $\mathcal{A}$ be given. Let s' and t' be as guaranteed by Theorem 4.1. We can assume that t' simulates s'. (That is, replacing t' by s' in any context gives at least as many root-to-root loops.) This can be enforced in a straightforward manner.

Let m be $|Q \times Q|$, the number of pairs of states of $\mathcal{A}$, and thus the number of different tree-loops of $\mathcal{A}$. For $i \geq 0$, let t_i denote the tree which has a leftmost branch of length $m+1$ which has s' and t' subtrees as right offspring. More precisely, a node of the form 0^j1 has s' as subtree if $j \leq i$ and otherwise t'. Clearly, t_i is in $\mathcal{L}_{\mathrm{even}}$ iff i is even. Note that t_{i+1} is obtained from t_i by replacing one subtree s' with t'. It is easy to see that therefore t_{i+1} has all root-to-root loops of $\mathcal{A}$ that t_i has. Thus, the t_i, for $0 \leq i \leq m+1$, induce a monotone sequence of $m+2$ sets of root-to-root loops and, consequently, there must be an i such that the sets induced by t_i and t_{i+1} are identical. We can choose one of them as s and the other as t. □

We now define the languages that will be used in our separation proofs. They all consist of trees of a certain shape. A tree is **n-leveled**, for $n \geq 0$, if each of its paths from the root to a leaf is labeled by a sequence of the form $(\mathbf{cb}^*)^n(\mathbf{a}+\mathbf{b})$. Thus, in an n-leveled tree the root is labeled with c, there are n antichains labeled by **c**, some leaves have label **a** and all the other nodes are labeled by **b**. Note that a 0-leveled tree consists of a single node labeled with **a** or **b**. A node is said to be **on level** i if its subtree is an i-leveled tree; it must therefore be labeled by **c**. We sometimes identify a level in a tree with the nodes of that level, each level thus forms a maximal antichain. The **level parent** of a node v is the closest ancestor of v that is on some level. A tree is **leveled** if it is n-leveled for some n.

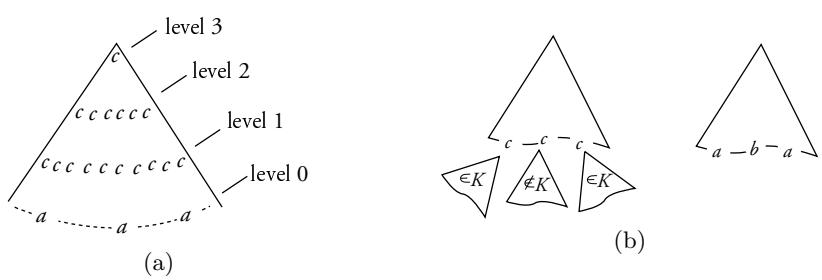


Fig. 2. Illustration of (a) a leveled tree, (b) a leveled tree and its folding.

For a language $\mathcal{K}$ of $(n-1)$-leveled trees, the **$\mathcal{K}$-folding** of an n-leveled tree t is defined as follows. The label of the root is set to **b**. All nodes below level $n-1$ are removed. Each node v at level $n-1$ is labeled by **a** if $t|_v \in \mathcal{K}$ and by **b** otherwise. The folding of a 0-leveled tree is just the tree itself with the root label set to **b**.

In the remainder of the section, we only consider leveled trees and their subtrees. Let languages $\mathcal{L}_0, \mathcal{L}_1, \ldots$ and $\mathcal{M}_0, \mathcal{M}_1, \ldots$ be defined as follows.

- $\mathcal{L}_0 = \mathcal{M}_0$ contains only the single node tree with label **a**.
- $\mathcal{L}_n$ is the set of all n-leveled trees whose $\mathcal{L}_{n-1}$-folding is in $\mathcal{L}_{\text{even}}$.
- $\mathcal{M}_n$ is the set of n-leveled trees whose $\mathcal{M}_{n-1}$-folding is in $\mathcal{L}_{3\text{l}}$.

Note that $\mathcal{L}_1 = \mathcal{L}_{\text{even}}$ and $\mathcal{M}_1 = \mathcal{L}_{3\text{l}}$.

Proposition 4.3. *For each $n \geq 1$, (a) $\mathcal{L}_n \in \mathrm{DPA}_n - \mathrm{PA}_{n-1}$, and (b) $\mathcal{M}_n \in \mathrm{TWA} - \mathrm{DPA}_{n-1}$.*

Proposition 4.3 (a) immediately implies Theorem 1.2. Likewise, Theorem 1.3 immediately follows from Proposition 4.3 (b). The lower bounds are shown in the following subsections. The upper bounds are shown by induction, the difficulty being the initial case which will be detailed in the full version.

4.2 Oracle automata

The general idea of the lower bound proofs of Propositions 4.3 is that once an $(n-1)$-pebble automaton drops a pebble in the top level of an n-leveled tree t, with the remaining $n-2$ pebbles it cannot check whether the subtree of a node at level $n-1$ is in $\mathcal{L}_{n-1}$ (resp., $\mathcal{M}_{n-1}$). Thus, whenever the automaton uses a pebble at a node v in the top level it is *blind* with respect to the properties of the nodes at level $n-1$. But it still can check properties of v that depend on the position of v in the unlabeled version of t. In this subsection, we formalize this intuition by the notion of *oracle automata* which are an extension of tree-walking automata by *structure oracles*. Then we show that Theorem 4.1 also holds for oracle automata.

A **structure oracle** $\mathcal{O}$ is a (parallel) deterministic bottom-up tree automaton [10] that is label invariant. That is, any two trees that have the same nodes get assigned the same state by $\mathcal{O}$. Therefore, a structure oracle is defined by its state space Q, an initial state $s_0 \in Q$ and a transition function $Q \times Q \to Q$. We write $t^{\mathcal{O}}$ for the state of $\mathcal{O}$ assigned to a tree t. This notation is extended to contexts: given a context C, $C^{\mathcal{O}} : Q \to Q$ is defined by $C^{\mathcal{O}}(q) = (C[t])^{\mathcal{O}}$, where t is some tree with $q = t^{\mathcal{O}}$. (All states are assumed reachable.)

For a tree t over some alphabet Σ, a node v of t, and a structure oracle $\mathcal{O}$, the **structural $\mathcal{O}$-information** about (t, v) is the pair

$$((C_{t,v})^{\mathcal{O}}, (t|_v)^{\mathcal{O}}) \in Q^Q \times Q \ .$$

It should be noted that the result of any unary query expressible in monadic second-order logic which does not refer to the label predicates can be calculated based on the structural $\mathcal{O}$-information for some $\mathcal{O}$ (and vice-versa). Since the only type of oracles we use in this paper are structure oracles, we just write oracle from now on.

An **oracle tree-walking automaton** is a tree-walking automaton $\mathcal{A}$ (with state set Q) extended by a structure oracle $\mathcal{O}$ (with state set P). The only difference to a usual tree-walking automaton is in the definition of the transition relation. It is of the form:

$$\delta \subseteq (Q \times (P^P \times P) \times \Sigma) \times (Q \times \{\epsilon, \uparrow, \swarrow, \searrow, \text{lift}, \text{drop}\}).$$

Whether a transition of $\mathcal{A}$ is allowed depends on the current state of $\mathcal{A}$, the label of the current node v and the structural $\mathcal{O}$-information about (t, v). Note that this generalizes tree-walking automata, since the structural information can include the type. The **size** of an oracle tree-walking automaton is defined as $|P| + |Q|$.

The following proposition generalizes Theorem 4.1 and Proposition 4.2 to oracle automata:

Proposition 4.4. *(a) For each deterministic oracle automaton, there are trees $s \in \mathcal{L}_{3l}$, $t \notin \mathcal{L}_{3l}$ that have the same root-to-root loops.*
(b) For each oracle automaton, there are trees $s \in \mathcal{L}_{\text{even}}$, $t \notin \mathcal{L}_{\text{even}}$ that have the same root-to-root loops.

4.3 The proof of the lower bounds

This subsection is devoted to the lower bound part of Proposition 4.3. To this end, let $n \geq 1$ and $\mathcal{A}$ be an $(n-1)$-pebble automaton with m states.

We will inductively construct trees s_i and t_i, $i = 1, \ldots, n$, such that, for each i, (1) s_i and t_i are i-leveled, (2) $s_i \in \mathcal{L}_i$, $t_i \notin \mathcal{L}_i$, and (3) s_i and t_i are $(i-1)^*$-equivalent. The base trees s_1 and t_1 are taken from the following lemma, which is an immediate consequence of Proposition 4.4.

Lemma 4.5. *For every k, there are 1-leveled trees $s_1 \in \mathcal{L}_1, t_1 \notin \mathcal{L}_1$ that have the same root-to-root loops for every nondeterministic oracle tree-walking automaton of size $\leq k$.*

Let s_1 and t_1 be the trees obtained by this lemma for k large enough, depending on $\mathcal{A}$ and n. (The exact constraints on k are stated in the proof of Lemma 4.6). For $i > 1$, s_i is obtained from s_1 by replacing every **a** leaf with s_{i-1} and every **b** leaf with t_{i-1}. The tree t_i is analogously obtained from t_1. It is immediate that s_i and t_i are i-leveled trees and that $s_i \in \mathcal{L}_i$ and $t_i \notin \mathcal{L}_i$.

The lower bound of Proposition 4.3 (a) follows directly from Lemma 3.2 and:

Lemma 4.6. *For each $i = 0, \ldots, n-1$, the trees s_{i+1} and t_{i+1} are i^*-equivalent.*

Proof. The proof is by induction on i. For the base case $i = 0$, we need to show that the trees s_1 and t_1 admit the same 0-loops, i.e. loops that do not use any pebbles. But this follows from Lemma 4.5, since it corresponds to loops of a tree-walking automaton without pebbles (we do not even need the oracle). Since Lemma 4.5 talks about root-to-root loops, and we want s_1 and t_1 to be equivalent in any context, we need k to be greater than the state space of any automaton recognizing a 0-behavior from Lemma 3.6.

Let thus $i \geq 1$. We assume that s_i and t_i are $(i-1)^*$-equivalent, we need to show that s_{i+1} and t_{i+1} are i^*-equivalent. An $(i+1)$-leveled tree where all i-leveled subtrees are either s_i or t_i is called **difficult**. Clearly both s_{i+1} and t_{i+1} are difficult. Let τ_s and τ_t be the i^*-behaviors of s_i and t_i, respectively. Note that τ_s and τ_t may be different, our induction assumption only says that the $(i-1)^*$-behaviors of s_i, t_i are the same. The **behavior folding** $\overline{t}$ of a difficult tree t is the i^*-behavior folding of t where every occurrence of t_i is replaced by a single node labeled with τ_t, similarly for s_i. Note that the behavior foldings of s_{i+1}, t_{i+1} are essentially the trees s_1, t_1, except that **a** is replaced by τ_s and **b** is replaced by τ_t.

Let B be a j-behavior, with $j \leq i$. In order to complete the proof of the lemma, we need to show that B is the j-behavior of s_{i+1} if and only if it is the j-behavior of t_{i+1}. Let $\mathcal{C}$ be the automaton from Lemma 3.7 that accepts i^*-foldings of trees with j-behavior B. We only consider the most difficult case, when $j = i$ and $\mathcal{C}$ has i pebbles. We will show that

Claim. $\mathcal{C}$ accepts the behavior folding of s_{i+1} iff it accepts the behavior folding of t_{i+1}.

The general idea is that over behavior foldings of difficult trees, the i-pebble automaton $\mathcal{C}$ can be simulated by an oracle tree-walking automaton. That is, we will construct an oracle tree-walking automaton $\mathcal{D}$ that accepts exactly the same behavior foldings of difficult trees as $\mathcal{C}$. The size of $\mathcal{D}$ will depend only on the size of $\mathcal{C}$ (and hence in turn, on the size of $\mathcal{A}$). The result follows, as long as the k used in defining s_1 and t_1 was chosen large enough so that $\mathcal{D}$ cannot distinguish the behavior foldings of s_{i+1} and t_{i+1} (which are the same as s_1, t_1).

We now proceed to show how the simulating oracle tree-walking automaton $\mathcal{D}$ is defined. Recall that an i-run of the automaton $\mathcal{C}$ in the behavior folding of a difficult tree t (actually in any tree) can be decomposed into a sequence of i-moves each of one of the following types:

– a single transition in which pebble i is not dropped on the tree;
– a *drop pebble* i transition, followed by an $i-1$-loop, followed by *lift pebble* i.

Clearly, a single transition of the former type can be simulated by a tree-walking automaton (even without any oracle). The remainder of this proof is to show how to simulate an i-move of the latter type. The following is the key claim:

Claim. Let v be a node in the behavior folding $\bar{t}$ of a difficult pebbled tree. Whether or not there is an $(i-1)$-loop from a state p to a state q in v does not depend on the labels of $\bar{t}$.

We now proceed to justify this claim. By the remark in the proof of Lemma 3.7, whether or not $\mathcal{C}$ admits an $(i-1)$-loop does not depend on all the information about the i^*-equivalence classes of s_i, t_i written in the leaves, but only on the information about $(i-1)^*$-equivalence. However, by our induction assumption all the $(i-1)^*$-equivalences written in the leaves are the same. The claim follows (recall that non-leaf nodes of $\bar{t}$ have the blank label).

The claim implies that $(i-1)$-loops of $\mathcal{C}$ on the behavior folding of a difficult tree can be simulated by an $(i-1)$-pebble automaton whose behavior does not depend on node labels. By translating this automaton into a parallel automaton, we can create an oracle $\mathcal{O}$ that provides at each node v the set of pairs (p, q) for which there is an $(i-1)$-loop at v. □

The proof of the lower bound of Proposition 4.3 (b) is completely analogous.

Proof (of Theorem 1.1). We will define a regular tree language $\mathcal{L}$ that is not recognized by any pebble automaton. Note that we can not use the union of all $\mathcal{L}_i$, since this language requires checking that all paths have the same number of **c** labels.

The general idea though, is the same: the intersection of $\mathcal{L}$ with the set of i-leveled trees will be exactly $\mathcal{L}_i$. In particular, all the trees s_i from the previous lemma belong to $\mathcal{L}$, but none of the trees t_i does. Therefore, no pebble automaton can recognize $\mathcal{L}$.

Now we define the language $\mathcal{L}$. Every path in every tree from $\mathcal{L}$ is of the form $(\mathbf{cb}^*)^*(\mathbf{a}+\mathbf{b})$. The tree with the single node **a** is in $\mathcal{L}$. Furthermore, a tree is in $\mathcal{L}$ if its $\mathcal{L}$-folding is in $\mathcal{L}_{\text{even}}$. Here, the $\mathcal{L}$-folding of a tree with paths of the form $(\mathbf{cb}^*)^*(\mathbf{a}+\mathbf{b})$ is obtained by replacing each node whose only **c** ancestor is the root by a leaf with **a** if its subtree is in $\mathcal{L}$, and by a leaf with **b** otherwise. This language clearly satisfies the desired properties.

□

We do not know if the language $\mathcal{M}$, analogously constructed from the $\mathcal{M}_i$, is in TWA. If it was we would get TWA $\not\subseteq$ DPA, and thus, by the result of [5], FO+DTC $\subsetneq$ FO+posTC.

5 Strong pebbles are weak

The goal of this section is to prove Theorem 1.4. Given a strong n-pebble automaton we construct an equivalent standard n-pebble automaton. As a means to prove this equivalence we introduce an intermediate model. An n-pebble automaton is **k-weak** if pebbles $1, \ldots, k$ are weak (and can be lifted only when the head is on them) and pebbles $k+1, \ldots, n$ are strong (and can be lifted from anywhere). We intend to prove the following lemma from which the nondeterministic part of Theorem 1.4 follows by induction.

Lemma 5.1. *For every $0 \leq k < n$, each k-weak n-pebble automaton $\mathcal{A}$ has an equivalent $(k+1)$-weak n-pebble automaton $\mathcal{A}'$.*

Proof. Let k, n be fixed and let $\mathcal{A}$ be a k-weak n-pebble automaton. Let π be a $(k+1)$-run of $\mathcal{A}$ between two placements of the pebble $k+1$. This run begins by dropping pebble $k+1$ at a node v resulting in a configuration $(v, p, v, v_{k+2}, \ldots, v_n)$. Then it continues with a k-run ρ ending in a configuration $(w, q, v, v_{k+2}, \ldots, v_n)$. Finally, pebble $k+1$ is lifted. By

$u_1 = v, \ldots, u_m = w$ we denote the nodes on the path in the tree from v to w. Since v, w need not be equal, π may violate the conditions imposed on $(k+1)$-weak automata.

In the following, we describe how a $(k+1)$-weak automaton $\mathcal{A}'$ can simulate π. We do only the case when v is an ancestor of w. The cases when w is an ancestor of v, or when v, w are incomparable, are similar.

We note first that, as in ρ pebble $k+1$ is never lifted, it is actually a k-run of a standard n-pebble automaton. Thus, we can use all the machinery developed in Section 3 to reason about ρ and its subruns.

We decompose ρ as $\rho_1, \ldots, \rho_m$, such that, for each $m \geq i \geq 2$, ρ_{i-1} is a context k-loop of $C_{t,u_i}[t|_{u_i}]$. Thus, for $i > 1$, the first time that node u_i is entered in a k-configuration is the first configuration of ρ_i. Let, for each i, p_i be the first state in ρ_i.

$\mathcal{A}'$ can simulate π as follows. First it guesses a context k^*-behavior B_1 and verifies that $B_1 \leq B^k_{C_{t,v}}$ as in Lemma 3.6.

Assume a context k^*-behavior B_i has been computed such that $B_i \leq B^k_{C_{t,u_i}}$ (here C_{t,u_i} refers to the context where pebble $k+1$ is still on the node u_i). Assume that $\mathcal{A}'$ and pebble $k+1$ are currently at u_i and that the current state is p_i. Then $\mathcal{A}'$ inductively proceeds as follows, for every $i \geq 1$. It first guesses whether w is in the left or the right subtree. Assume it guessed that it is in the left subtree. Thus, u_{i+1} is a left child of u_i. Let a_i be the label of u_i, R_i its pebble set, t_i be $t|_{u_i}$ without pebble $k+1$ and t'_i be the right subtree of u_i. $\mathcal{A}'$ guesses the $(k-1)^*$-equivalence class τ_i of t_i and the t^*-equivalence class τ'_i of t'_i and, using Lemma 3.6, checks whether t_i and t'_i are in a class bigger than τ_i and τ'_i. If this is the case it guesses a state q_i and verifies that $(p_i, q_i) \in B_i(\tau_i)$. By Lemma 3.2 this guarantees that (p_i, q_i) is a k-loop in C_{t,u_i}. Let B_{i+1} be the k^*-behavior corresponding to $\mathbf{Compose}(B_i, a_i, R_i, *, \tau'_i)$ (which is a correct notation by Lemma 3.5). By Lemma 3.5 we have $B_{i+1} \leq B^{k+1}_{C_{t,u_{i+1}}}$. $\mathcal{A}'$ can now safely move pebble $k+1$ from u_i to u_{i+1}, and simulates $\mathcal{A}$ in order to move from u_i in state q_i to u_{i+1} in state p_{i+1}.

At w we also need to simulate $\mathcal{A}$ on the tree k-loop of ρ_m and then we lift pebble $k+1$. $\square$

The deterministic case requires a slightly more care but essentially follows the same idea.

Lemma 5.2. *For every $k < n$, each k-weak pebble deterministic automaton $\mathcal{A}$ with n pebbles has an equivalent $(k+1)$-weak pebble deterministic automaton $\mathcal{A}'$ with n pebbles.*

References

1. A. V. Aho, J. D. Ullman Translations on a Context-Free Grammar. In *Information and Control*, 19(5): 439-475, 1971.
2. M. Bojańczyk and T. Colcombet. Tree-Walking Automata Cannot Be Determinized. TCS, to appear.
3. M. Bojańczyk and T. Colcombet. Tree-walking automata do not recognize all regular languages. In *STOC*, 2005.
4. J. Engelfriet and H.J. Hoogeboom. Tree-walking pebble automata. In *Jewels are forever*, (J. Karhumäki et al., eds.), Springer-Verlag, 72-83, 1999.
5. J. Engelfriet and H.J. Hoogeboom. Nested Pebbles and Transitive Closure. In *STACS*, 2006.
6. J. Engelfriet, S. Maneth. A comparison of pebble tree transducers with macro tree transducers. In *Acta Inf.* 39(9): 613-698, 2003.
7. J. Engelfriet, H.-J. Hoogeboom, J.-P.Van Best. Trips on Trees. In *Acta Cybern.* 14(1): 51-64, 1999.
8. T. Milo, D. Suciu and V. Vianu. Typechecking for XML transformers. In *J. Comput. Syst. Sci.*, 66(1): 66-97, 2003.
9. A. Muscholl, M. Samuelides and L. Segoufin. Complementing deterministic tree-walking automata. In *IPL*, to appear.
10. H. Comon et al. Tree Automata Techniques and Applications. Available at http://www.grappa.univ-lille3.fr/tata