Complementing deterministic tree-walking automata

Anca Muscholl
LIAFA & CNRS, Université Paris 7, France

Mathias Samuelides
LIAFA and INRIA

Luc Segoufin
INRIA & Université Paris 11, France

September 12, 2005

Abstract

We consider various kinds of *deterministic* tree-walking automata, with and without pebbles, over ranked and unranked trees. For each such kind of automata we show that there is an equivalent one which never loops. The main consequence of this result is the closure under complementation of the various types of automata we consider with a focus on the number of pebbles used in order to complement the automata.

1 Introduction

On trees, there are two types of automata that extend automata on strings in a natural way. The best known automaton type has a parallel (branching), one-way behavior: a run of the automaton is a labeling of the tree by states, either in top-down or bottom-up fashion, and according to local update rules. Branching automata appear in several variants, besides bottom-up and top-down they can be deterministic or non-deterministic. Except for the deterministic top-down variant, the other three have all the nice properties of the string case, namely closure under Boolean operations, determinization (of the bottom-up variant), minimization, equivalence with monadic-second order logic, etc. These automata are commonly called *tree automata* and the family of tree languages they define is called the class of *regular tree languages*. See for instance the textbook [CDG$^+$99] for more details.

In this paper we are interested in a second variant, namely *tree-walking automata* (TWA for short), which generalize two-way string automata. Given the state the automaton has reached at a given node and the label of that node, it switches to a new state and moves in the tree to a neighboring node according to the transition function. It accepts when it reaches an accepting state. Thus, this variant is sequential, in the sense that the automaton head is positioned on a single node. TWA have a non-deterministic (NTWA) and a deterministic variant (DTWA). They have been introduced in [AU71] and it is not hard to see that languages recognized by TWA are regular. It has been shown only very recently that the inclusion is strict [BC05]. It has also been shown that NTWA are strictly more powerful than DTWA [BC04].

A *pebble tree-walking automaton* (PTWA for short) is an extended TWA which can drop and lift a fixed number of pebbles in the tree. In order to stay within regular tree languages the pebbles are constrained by a stack discipline [GH96, EH99]. That is, pebbles are ordered, pebble i can be lifted from the tree only if there is no pebble $j > i$ present on the tree and we can drop only the smallest available pebble. We consider two variants of PTWA. In the *weak* setting (wPTWA for short), which is the most studied one (see for instance [EH99] and [MSV03]), the last pebble can be lifted only if the automaton head is currently on the corresponding node. In the *strong* setting (sPTWA for short) the automaton can lift the last pebble from any node. This last variant has been introduced in [EH05] and corresponds to a robust class of automata, as it captures unary transitive closure logic on trees [EH05]. In both cases of wPTWA and sPTWA the recognized tree languages are regular since unary transitive logic can be simulated by monadic second-order logic.

The interest for PTWA has recently increased because of its connections with query languages for XML. For instance they can be used to evaluate XPATH queries. A pebble is then used for each qualifier occurring in the query. It has also been advocated in [MSV03] that PTWA have all the navigational power of many of the existing query languages for XML. This was used in [MSV03] in order to decide typechecking properties for XML query languages.

It is still open whether NTWA and (w/s)NPTWA are closed under complementation or whether the inclusion of (w/s)PTWA in the class of regular tree languages is strict or not.

In this paper we show that all *deterministic* variants (DTWA, wDPTWA and sDPTWA) are closed under complementation.

Recall that the acceptance condition of these automata is that an accepting state is eventually reached. Therefore there are two reasons for rejecting a tree. Either a rejecting state is reached or the automaton loops forever. This second case is problematic when one wants to compute an automaton for the complement language. We show that for any DTWA $\mathcal{A}$ there is an equivalent DTWA $\mathcal{A}'$ which never loops. We then extend this result to wDPTWA by showing that for any k-pebble wDPTWA there is an equivalent k-pebble wDPTWA which never loops. Finally we consider sDPTWA but in that case $3k$ pebbles are used in order to remove all loops of a k-pebble sDPTWA. It remains open whether this blow-up is unavoidable or not.

The idea of the proof is based on Sipser's crucial observation [Sip78] that the *backward configuration graph* of a deterministic Turing Machine M on any of its input w is a forest. The vertices of this graph are the configurations of M on w, and an edge $c \to c'$ connects two configurations c and c' if c can be reached from c' in one step of M. In order to remove loops, it is therefore enough to be able to simulate M on w backwards, starting from an accepting configuration and checking all possible paths reaching this accepting configuration until an initial configuration is reached. This is done by investigating the configuration tree e.g. in a depth-first-search fashion.

The difficulty is to be able to do the simulation with the limited power of finite automata. Indeed, the backward configuration tree cannot be stored in the finite control of the automaton. But it turns out that the tree can be computed locally and on-the-fly, and this idea has been used by Sipser to solve the 2-way string case [Sip78].

We show that this idea easily extends to tree-walking automata. It also extends to wDPTWA, where the moves of the pebbles are local. For the sDPTWA variant, backwards simulation is a bit more complex. Indeed, in that case an automaton $\mathcal{A}$ can lift the last pebble from anywhere. Therefore, when simulating $\mathcal{A}$ backwards, the previous position of the pebble before a lift must be checked over the whole tree. We show that this can be done still deterministically, but with the help of extra pebbles.

Note that the closure under complement of a DTWA was mentioned in section 4 of [EH99]. Note also that the closure under complement of an sDPTWA immediately follows from the fact that sDPTWA captures precisely unary deterministic transitive closure logic [EH05], the latter being closed under complement by definition. But this approach does not yield a very efficient procedure in term of pebbles. From [EH05] it follows that the complementation of a sDPTWA using k pebbles and n states is expressible using a sDPTWA with $9kn$ pebbles, while our construction uses only $3k$ pebbles.

2 Notations and TWA

We denote the size of a finite set E by $|E|$. The trees we deal with in this section are finite binary trees, with nodes labeled over the alphabet Σ. A Σ-tree t is a mapping from $N_t \subseteq \{0,1\}^*$ to Σ where N_t is a finite, non empty, prefix-closed set such that for any $v \in N_t$, $v0 \in N_t$ iff $v1 \in N_t$. We use the set Types $= \{\epsilon, \epsilon_l, l_0, l_1, ch_0, ch_1\}$ to encode the possible types of a node: the root ϵ, a left-child leaf l_0, a right-child leaf l_1, a leaf and root at the same time ϵ_l, a left-child inner node ch_0 or a right-child inner node ch_1. For $v \in N_t$, let $type_t(v)$ denote the type of the node v in the tree t. Let $d : N_t \times N_t \to \{\uparrow, stay, \downarrow_0, \downarrow_1\}$ be the partial function assigning $\downarrow_i$ to pairs of the form (v, vi) for $i \in \{0,1\}$, *stay* to pairs of form (v,v) and $\uparrow$ to pairs of the forms (vi, v), for $i \in \{0,1\}$. We denote the depth-first traversal of a tree (that is, the traversal visiting at each node, the left subtree first and then the right subtree) by **DFS** and by right-left DFS the traversal that visits first the right subtree before the left one.

Definition 1 *A **tree-walking automaton** (TWA) over Σ-trees is a tuple $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states and $\delta \subseteq Q \times Types \times \Sigma \times Q \times \{\uparrow, stay, \downarrow_0, \downarrow_1\}$ is the transition function. A TWA is called* deterministic *(DTWA) if δ is a function from $Q \times Types \times \Sigma$ to $Q \times \{\uparrow, stay, \downarrow_0, \downarrow_1\}$.*

A *configuration* of $\mathcal{A}$ over a Σ-tree t is a pair $(q,v) \in Q \times N_t$ indicating the current state and the position of the head in the tree. We write $(q,v) \xrightarrow{\mathcal{A},t} (q',v')$ for $(q, type_t(v), t(v), q', d(v,v')) \in \delta$.

A *run* of $\mathcal{A}$ over a Σ-tree t is a sequence of configurations $(q_0,v_0),\ldots,(q_n,v_n)$ satisfying for all $0 \le i < n$, $(q_i,v_i) \xrightarrow{\mathcal{A},t} (q_{i+1},v_{i+1})$.

A run can be finite or infinite. A run $(q_0,v_0)\ldots(q_n,v_n)$ is *accepting* if it is finite and it starts at the root in q_0 and *ends at the root* in a final state. A TWA $\mathcal{A}$ accepts a tree if it has an accepting run over it. A set of Σ-trees L is recognized by $\mathcal{A}$ if $\mathcal{A}$ accepts exactly the trees in L.

We consider now a DTWA $\mathcal{A}$ and would like to construct a DTWA for the complement of $\mathcal{A}$. A DTWA is called *non-looping* if every run starting from the initial configuration is finite. For non-looping DTWA, complementation is immediate as it is enough to tranform a accepting state into a rejecting one and vice-versa. The difficulty for obtaining complementation comes from those loops.

In order to complement $\mathcal{A}$, we construct a DTWA $\mathcal{A}'$ that recognizes the same language as $\mathcal{A}$ and such that all runs in $\mathcal{A}'$ are finite.

Proposition 1 *For any DTWA with n states we can construct an equivalent, non-looping DTWA with $O(n^2)$ states.*

From Proposition 1 we obtain immediately:

Theorem 1 *The class of tree languages recognized by DTWA is closed under complementation.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ be a DTWA that accepts the set L. We construct a DTWA $\mathcal{A}'$ that accepts the complement of L as follows. According to Proposition 1 we can assume that all runs of $\mathcal{A}$ starting in the initial configuration are finite and end at the root. We introduce a new state q'_f that is the final state of $\mathcal{A}'$. Then for each non final state $q \in Q \setminus F$ of $\mathcal{A}$ and for each letter a such that $\delta(q, \epsilon, a)$ is not defined, we add the transition $\delta'(q, \epsilon, a) = (q'_f, stay)$. $\Box$

We now show Proposition 1. We start with a normal form for DTWA that will simplify the case analysis.

Lemma 1 *For every DTWA an equivalent DTWA with a unique final state q_f can be constructed, such that the unique final configuration (q_f, ϵ) has no successor configuration.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ be a DTWA. We introduce a new state q'_f that is the unique final state of $\mathcal{A}'$. Then we suppress all transitions that can be applied from a configuration from $F \times \{\epsilon\}$. For each state q_f in F and for each letter $a \in \Sigma$, transition $\{(q_f, \epsilon, a, q'_f, \mathit{stay})\}$ is added. □

From now on, we will consider only DTWA that satisfy the condition of the previous lemma and we will denote a DTWA as $(Q, \Sigma, q_0, q_f, \delta)$ instead of $(Q, \Sigma, q_0, \{q_f\}, \delta)$. We now define the backward configuration graph of a DTWA. Let $\mathcal{A} = (Q, \Sigma, q_0, q_f, \delta)$ be a DTWA and let t be a Σ-tree.

Definition 2 *The* ***backward configuration graph*** $G(\mathcal{A}, t)$ *is the finite graph whose vertices are the configurations c of $\mathcal{A}$ over t such that there exists a run from c to the final configuration (q_f, ϵ) and there is an edge $c \to c'$ iff $c' \xrightarrow{\mathcal{A},t} c$.*

Note that a tree t is accepted by $\mathcal{A}$ iff the initial configuration of $\mathcal{A}$ is a vertex of $G(\mathcal{A}, t)$. The following lemma is a crucial remark:

Lemma 2 ([Sip78]) *Let $\mathcal{A} = (Q, \Sigma, q_0, q_f, \delta)$ be a DTWA. For all trees t, the backward configuration graph $G(\mathcal{A}, t)$ is a tree with root (q_f, ϵ).*

Proof. From the definition of the backward configuration graph, all vertices of $G(\mathcal{A}, t)$ are reachable from (q_f, ϵ) and for any vertex (q, v) there is a unique path from (q_f, ϵ) to (q, v), since $\mathcal{A}$ is deterministic and (q_f, ϵ) is not a successor of any vertex of $G(\mathcal{A}, t)$. □

In order to eliminate the infinite (non-accepting) runs of $\mathcal{A}$ we construct a DTWA $\mathcal{A}'$ that simulates $\mathcal{A}$ *backwards*. On a given tree t, $\mathcal{A}'$ performs a DFS of the backward configuration tree $G(\mathcal{A}, t)$ from the final configuration of $\mathcal{A}$, that is from the root of $G(\mathcal{A}, t)$, and accepts t iff it eventually visits the initial configuration. Note that the backward configuration tree is implicit, i.e. the DFS is simulated on-the-fly during the traversal of t. This can be done because the backward configuration tree is *locally constructible*: we can compute from a given configuration all its successors and its unique predecessor in the backward configuration tree.

Construction. Let $\mathcal{A} = (Q, \Sigma, q_0, q_f, \delta)$ be a DTWA and t a tree. Let $D : N_t \times N_t \to \{\uparrow_0, \uparrow_1, \mathit{stay}, \downarrow_0, \downarrow_1\}$ be the partial function assigning $\uparrow_i$ to pairs of the forms (vi, v), for $i \in \{0, 1\}$ and the result of d otherwise. Given a node (q, v) of $G(\mathcal{A}, t)$ with parent node (q', w), we define its *rank* by the pair $\langle q, D(v, w)\rangle$. The basic observation is that the rank of a configuration node n uniquely determines n among its siblings in the backward configuration tree. We fix now an arbitrary order on Q and on the set $\{\uparrow_0, \uparrow_1, \mathit{stay}, \downarrow_0, \downarrow_1\}$. This implies a total order on the rank (lexicographically) and therefore a total order on the children of any node in the backward configuration tree. The DTWA $\mathcal{A}'$ will perform a DFS of $G(\mathcal{A}, t)$ according to this order. Whenever $\mathcal{A}'$ is visiting a configuration (q, v) of $G(\mathcal{A}, t)$ in the DFS, the head of $\mathcal{A}'$ is visiting the node v of t, and the state q is part of the current state of $\mathcal{A}'$. The current state of $\mathcal{A}'$ also always contains the rank of the node previously visited in order for $\mathcal{A}'$ to know which node it should go to next.

This information is maintained as follows. Assume that $\mathcal{A}'$ is currently investigating the node $n = (q, v)$ of $G(\mathcal{A}, t)$. Its head is therefore on v and q is part of the state.

There are two cases. Assume first that the node n is visited for the first time. This is the case exactly when n is reached from its parent node in $G(\mathcal{A}, t)$. Then $\mathcal{A}'$ determines the rank of the first child of n as follows: $\mathcal{A}'$ tries each rank $\langle q', \Delta' \rangle$ in increasing order. For each one it moves to the node w in direction opposite to Δ' and (virtually) checks using δ and the label a of w whether $\delta(q', a) = q, \Delta'$. If this is not the case, it goes back to v (with state q) and tries the next rank.

The second case is when the DFS returns to node $n = (q, v)$ from a child n' of n of rank $\langle q', \Delta' \rangle$. This rank is thus part of the current state of $\mathcal{A}'$. If $\langle q', \Delta' \rangle$ is the maximal element in the rank order then $\mathcal{A}'$ returns to the parent of n in $G(\mathcal{A}, t)$. To do this, $\mathcal{A}'$ simulates virtually $\mathcal{A}$ on t for one step and obtains a new state r and a direction Δ. It moves according to Δ and maintains in its state the necessary information: r for the state of the current configuration and $\langle q, \Delta \rangle$ for the rank of the previously visited node n.

If $\langle q', \Delta' \rangle$ is not maximal, $\mathcal{A}'$ proceeds to the next child of n by investigating all the next possibilities for rank as described above.

Note that the size of *rank* is bounded by $5|Q|$ therefore the extra information stored in the states of $\mathcal{A}'$ is bounded by $5|Q|^2$. We also need an extra bit in order to know whether the DFS goes downwards or upwards. Overall the number of states of $\mathcal{A}'$ is $O(|Q|^2)$.

3 Extensions

In this section we consider two extensions of Theorem 1, the first one over unranked trees and the second one for pebble tree-walking automata.

3.1 Unranked trees

A deterministic tree-walking automaton over unranked trees (DTWA$_U$ for short) is a DTWA that runs on unranked trees.

In this context we slightly modify the meaning of the type of a node. Recall that the set of types is Types $= \{\epsilon, \epsilon_l, l_0, l_1, ch_0, ch_1\}$. Their meaning on unranked trees is: the root ϵ, a leftmost child leaf l_0, a rightmost child leaf l_1, a leaf and root at the same time ϵ_l, a leftmost child inner node ch_0 or a rightmost child inner node ch_1. The set of moves, $\{\uparrow, \mathit{stay}, \downarrow_0, \downarrow_1 \leftarrow, \rightarrow\}$, should now be understood as: move to the parent of the current node, stay in the current node, move to the leftmost or rightmost child, move to the next or previous sibling.

We distinguish two models of automata, depending whether it also knows the label of the parent of the current node or not. In the first case δ maps $Q \times \text{Types} \times \Sigma \times \Sigma$ to $Q \times \{\uparrow, \mathit{stay}, \downarrow_0, \downarrow_1, \leftarrow, \rightarrow, \}$, in the second case δ is a function from $Q \times \text{Types} \times \Sigma$ to $Q \times \{\uparrow, \mathit{stay}, \downarrow_0, \downarrow_1, \leftarrow, \rightarrow\}$. These two models seem to differ, for instance a DTWA$_U$ of the first kind can accept all boolean circuits that evaluate to 1 in the usual way, by checking that all children of and-nodes (at leats one child of or-nodes, resp.) evaluate to 1. This obvious algorithm does not work if the automaton is forced to visit the parent node in order to know whether it is an and-node or an or-node, since it cannot record which child it came from. Anyway, our complementation algorithm does not depend on the model.

Note first that we cannot extend directly the previous construction to DTWA$_U$ because it is not clear how to define a *rank* which uniquely determines a node of $G(\mathcal{A}, t)$ from its siblings with a constant memory information. Indeed, when $\mathcal{A}$ moves up in the input tree it can do so from any child. Therefore in the backward configuration graph $G(\mathcal{A}, t)$ a node may have arbitrarily many children of rank $\langle q, \uparrow \rangle$.

However we can transform a DTWA$_U$ into an equivalent one, for which the backward configuration tree has finite rank. The new DTWA$_U$ can move upwards in a tree only when it is in a leftmost child node. We can enforce this property as follows: instead of going directly upward from an arbitrary son v of u to its father u, $\mathcal{A}'$ goes to the leftmost sibling of v and then goes to u.

In this case the *rank* of a configuration is defined as in the DTWA case with the obvious definition of D and it uniquely determines the siblings of a node in $G(\mathcal{A}, t)$ with a constant memory information. We can therefore apply the previous construction and obtain:

Theorem 2 *The class of tree languages recognized by DTWA$_U$ is closed under complementation.*

3.2 Pebble automata

We now formally define TWA with pebbles (PTWA). For simplicity, we define PTWA that use a stack discipline on the pebbles, since we are mainly interested in this type of automaton (without such a constraint on pebbles we can define non regular tree languages, see e.g. [GH96]).

Definition 3 *Let $k \geq 0$. A k-pebble PTWA is a tuple $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states and δ is the transition function:*

$$\delta \subseteq Q \times \textit{Types} \times \Sigma \times \{0, \ldots, k\} \times \{0,1\}^k \times Q \times \{\uparrow, \textit{stay}, \downarrow_0, \downarrow_1, \textit{lift}, \textit{drop}\}$$

A PTWA is deterministic (DPTWA), if δ is a function from $Q \times \textit{Types} \times \Sigma \times \{0, \ldots, k\} \times \{0,1\}^k$ to $Q \times \{\uparrow, \textit{stay}, \downarrow_0, \downarrow_1, \textit{lift}, \textit{drop}\}$.

In this setting each pebble is assigned a number and only the highest one can be lifted or droped. The additional components in the transition function of a DPTWA are the number of pebbles currently present in the tree (integer from $\{0, \ldots, k\}$) and the presence of pebble i at the current node (boolean vector from $\{0,1\}^k$); the additional moves can *lift* the last pebble present in the tree or *drop* the next available one, according to a given strategy. A configuration of a PTWA is a tuple $c = (q, v, i, \bar{x}) \in Q \times N_t \times \{0, \ldots, k\} \times N_t^k$ describing the current state q, the current node v, the current number of pebbles i and their positions $x_1, \ldots, x_i$ in the tree.

We describe now the *lift* and the *drop* move, the others being identical to the moves of a TWA. The *lift* move from a configuration $c = (q, v, i, \bar{x})$ yields the new configuration $c' = (q', v', i-1, \bar{x}')$ where $x'_j = x_j$ for all $j < i$. Moreover, we require that $v' = v$, i.e., the head does not move. The *drop* move from a configuration $c = (q, v, i, \bar{x})$ yields the new configuration $c' = (q', v', i+1, \bar{x}')$, where $x'_j = x_j$ for all $j \leq i$ and $x_{i+1} = v$. Again, we require that $v = v'$.

Without any further restriction, this defines *strong PTWA* (sPTWA). The most studied type of pebble automata has one further restriction [GH96, EH99, MSV03]: The *lift* move is further restricted as follows. We allow a *lift* move only from configurations $c = (q, v, i, \bar{x})$ with $v = x_i$, that is, the head is currently on the highest pebble present in the tree. This kind of PTWA is called *weak PTWA* (wPTWA).

Let $\mathcal{A}$ be a k-pebble wDPTWA. We construct as in the DTWA case a non looping k-pebble wDPTWA $\mathcal{A}'$ which recognizes the same language as $\mathcal{A}$ by simulating $\mathcal{A}$ backwards. Whenever $\mathcal{A}'$ will be visiting a node $(q, v, i, \bar{x})$ of $G(\mathcal{A}, t)$ it will have q in its state and its head will be on node v of t with the i pebbles placed according to $\bar{x}$. The rank of a configuration is defined as in the DTWA case with D mapping pairs of consecutive configurations to the set $\{\uparrow_0, \uparrow_1, \textit{stay}, \downarrow_0, \downarrow_1, \textit{lift}, \textit{drop}\}$ depending on the corresponding transition of $\mathcal{A}$. Because the head does not move when we lift or drop a pebble, the rank of a configuration uniquely determines a node among its siblings in $G(\mathcal{A}, t)$ and it can be stored with constant memory. Therefore we can apply the construction of Proposition 1 and have:

Theorem 3 *Let $k \geq 0$. The class of tree languages recognized by k-pebble wDPTWA is closed under complementation.*

This idea no longer works for sDPTWA. Indeed, in that case the last pebble can be lifted from anywhere, therefore there can be arbitrarily many configurations of rank $\langle q, \textit{lift}\rangle$ that can reach a given configuration, depending on where the pebble was placed in the tree before the *lift*.

Let $\mathcal{A}$ be an sDPTWA and t a tree. In order to uniquely determine a configuration of this kind we need to know where the pebble was on t at the moment when it is lifted: given a node $n = (q, v, i, \bar{x})$ of the backward configuration tree $G(\mathcal{A}, t)$, with parent node $n' = (q', w, j, \bar{y})$, we define its *extended rank* by the triple $\langle q, D(n, n'), x_i\rangle$ where D is defined as before for wDPTWA and x_i is the position of the last pebble i in t (note that x_i is only needed when $D(n, n')$ is a *lift*). The extended rank uniquely defines a node among its siblings in $G(\mathcal{A}, t)$ but it can no longer be stored with constant memory.

As before we fix an order among the children in $G(\mathcal{A}, t)$ of a node $n = (q', w, j, \bar{y})$ according to their extended rank. This order is relative to w. It is based on an arbitrary order on Q, an arbitrary order on $\{\uparrow_0, \uparrow_1, \textit{stay}, \downarrow_0, \downarrow_1, \textit{lift}, \textit{drop}\}$ where *lift* is maximal, and an order on the nodes of t. For technical reasons that will become clear later we fix the following order among the nodes of t, which is relative to w: let $v < v'$ if during the DFS on t starting in w, node v is visited before v'.

We will use in the (backward) simulation some stages where $\mathcal{A}$ is simulated forwards between two distinguished nodes. In order to make this formal, we use two special colors a and b (that will correspond to the presence of certain pebbles) and let T^{ab} be the set of trees containing exactly one node n_a of color a and one node n_b of color b (both distinguished nodes may be equal). If q, q' are states of $\mathcal{A}$ and n, n' are nodes in a tree t then we denote by (q, n) the configuration of $\mathcal{A}$ on t where no pebble are present in t, and by $(q, n) \xrightarrow{\mathcal{A},t} (q', n')$ the fact that $\mathcal{A}$ can move in t from the configuration (q, n) to the configuration (q', n'). We prove by induction on k that:

Lemma 3 *Let $\mathcal{A}$ be a k-pebble sDPTWA and r, r' be two states of $\mathcal{A}$. There exists a non-looping $3k$-pebble sDPTWA $\mathcal{A}'$ with two distinguished states q'_0, q'_f such that for all $t \in T^{ab}$, $(r, n_a) \xrightarrow{\mathcal{A},t} (r', n_b)$ iff $(q'_0, n_a) \xrightarrow{\mathcal{A}',t} (q'_f, n_b)$.*

Proof. The base case $k = 0$ is shown exactly as in the construction of Proposition 1 by replacing $G(\mathcal{A}, t)$ with $G^{ab}_{rr'}(\mathcal{A}, t)$, which is the backward configuration tree of $\mathcal{A}$ on t with root (r', n_b) instead of (q_f, ϵ) and accepting node (r, n_a) instead of (q_0, ϵ). As n_b is distinguished, the automaton starts from n_a by positioning its head on n_b with state r', then it performs the DFS on $G^{ab}_{rr'}(\mathcal{A}, t)$ from here as in the proof of Proposition 1. As n_a is also distinguished, $\mathcal{A}'$ accepts and proceeds to n_b as soon as it reaches n_a with state r.

Let $k > 0$. We construct $\mathcal{A}'$ from $\mathcal{A}$ by constructing a sDPTWA that performs a DFS on $G^{ab}_{rr'}(\mathcal{A}, t)$ according to the order defined above. The difficulty is to be able to go, for *lift* moves, from one configuration $(q, v, i, \bar{x})$ to its sibling configuration $(q', v, i, \bar{y})$ where y_i is the successor of x_i in the order defined above. Indeed the automaton needs to be able to transfer the pebble i from x_i to y_i while staying in v (for *lift* moves).

Assume $\mathcal{A}'$ is currently visiting the node $n = (q, v, i, \bar{x})$ of $G^{ab}_{rr'}(\mathcal{A}, t)$. Then its head is on v, its state contains q and the rank (not the extended rank, which is not finite) of the node it has previously visited, and for each $j \in \{1, \ldots, i\}$ the pebbles $(3j-2)$, $(3j-1)$ and $3j$ are on the node x_j. This information is maintained as follows.

The DFS is continued as in the DTWA case by checking whether all children of a n in $G^{ab}_{rr'}(\mathcal{A}, t)$ have been visited. If not, then $\mathcal{A}'$ computes the next possible value for the extended rank as follows: If neither the current rank nor its successor is a *lift*, then $\mathcal{A}'$ computes directly the next possible extended rank and virtually checks whether the new configuration gets back to n when simulating $\mathcal{A}$ as in the DTWA case. If the current rank is not a *lift* but its successor is a *lift*, then $\mathcal{A}'$ drops the pebbles $3i-2$, $3i-1$ and $3i$ on v and virtually checks that the new configuration with $x_i = v$ gets back to n as will be described in the next case. If yes it proceeds the DFS, if not it checks the next possible extended rank.

Assume now that the last investigated extended rank corresponding to a configuration $n_1 = (q_1, v, i, \bar{y})$ and the next possible extended rank corresponding to a configuration $n_2 = (q_2, v, i, \bar{z})$ both correspond to a *lift*. Note that for $j < i$ we have $x_j = y_j = z_j$. In this case the pebbles $3i-2$, $3i-1$ and $3i$ are already placed on node y_i of t and $\mathcal{A}'$ proceeds as follows: $\mathcal{A}'$ lifts both pebbles $3i$ and $3i-1$ and drops $3i-1$ on v in order to know where to come back. Note that at this point pebble $3i-2$ can no longer be lifted. It then searches t in DFS (from v onwards) until it reaches pebble $3i-2$ on y_i. It then moves to the successor z_i of y_i in the order on t relative to v that has been defined earlier. It drops pebble $3i$ on z_i. The problem is now to come back to v (determined by the position of pebble $3i-1$) after placing all three pebbles $3i$, $3i-1$, $3i-2$ on z_i.

To do this $\mathcal{A}'$ simulates $\mathcal{A}$ (forwards) starting from all configurations of the form $(q'', z_i, i, \bar{z})$ for some q'', and checks that the first time that pebble i is lifted the head is on v with state q (that is on pebble $3i-1$). If this is not the case for any such configuration, then $\mathcal{A}'$ safely ignores the extended rank corresponding to n_2 and proceeds to the next available one by going back to pebble $3i$ and moving this one to the next node according to the order on t relative to v. If the check succeeds, then $\mathcal{A}'$ goes back to node z_i (that is on pebble $3i$), lifts pebbles $3i$, $3i-1$ and $3i-2$, and drops them all on z_i and then simulates $\mathcal{A}$ again (forwards) until it does a *lift* of pebble i. At this stage, $\mathcal{A}$ is back on node v and can proceed its investigation of $G^{ab}_{rr'}(\mathcal{A}, t)$.

The remaining difficulty is that $\mathcal{A}$ may loop and therefore it is unsafe to simulate $\mathcal{A}$ forwards. However notice that $\mathcal{A}$ needs only to be simulated from the distinguished node containing pebble $3i$ to the distinguished node containing pebble $3i-1$ and that along this computation, $\mathcal{A}$ never drops or lifts pebble i, nor the ones below (stack discipline). We fix i new colors $c_1 \cdots c_i$ and construct $\mathcal{A}_i$ from $\mathcal{A}$ as follows. We remove all transitions dropping and lifting pebbles j with $j \leq i$, we rename pebble $j > i$ by pebble $j-i$, and we transform any transition assuming the presence of a pebble $j \leq i$ by a transition assuming the color c_j at the current node. A node of the input tree of $\mathcal{A}_i$ has the new color c_j, $j \leq i$, if the current pebble $3j$ is present on that node. By construction $\mathcal{A}_i$ is a $k-i$-pebble sDPTWA and we apply Lemma 3 by induction with the distinguished initial node z_i (pebble $3i$) and terminal node v (pebble $3i-1$) and states q'' and q. We thus obtain an equivalent non-looping sDPTWA $\mathcal{A}'_i$ using $3(k-i)$ pebbles that is used instead of $\mathcal{A}$ to simulate it forwards in the above.

The last case is where all children of n have been visited and we move upwards in $G^{ab}_{rr'}(\mathcal{A}, t)$ with a *lift* move. Here, $\mathcal{A}'$ goes to the parent node as in the wDPTWA case, and it can safely lift all three pebbles $3i$, $3i-1$, and $3i-2$ as they are no longer needed. $\Box$

Applying Lemma 3 with the root node for both distinguished nodes immediately yields:

Theorem 4 *Let $k \geq 0$. The class of tree languages recognized by sDPTWA is closed under complementation. More precisely, for every k-pebble sDPTWA $\mathcal{A}$ there exists a $3k$-pebble sDPTWA that recognizes the complement of $L(\mathcal{A})$.*

4 Conclusion

We have succeeded in complementing k-pebble sDPTWA using $3k$ pebbles. It would be interesting to know whether this can be done using only k pebbles or whether $3k$ is really needed. Note however that it is not even known whether the "strong" model is actually stronger than the "weak" one. Indeed there is no evidence that sDPTWA accept more tree languages than wDPTWA. One can show that for $k = 1$ the strong model collapses to the weak one. But this does not seem to easily extend to $k > 1$. Note that this implies that complementing a 1-pebble sDPTWA can be achieved using only *one* pebble.

Thanks to Thomas Schwentick, Joost Engelfriet and Hendrik Jan Hoogeboom for useful discussions on the topic, and to the referees for their suggestions for improvement.

References

[AU71] Alfred V. Aho and Jeffrey D. Ullman. Translations on a context-free grammar. *Information and Control*, 19(5):439–475, 1971.

[BC04] Mikolaj Bojanczyk and Thomas Colcombet. Tree walking automata cannot be determinized. In *Proc. of Intl. Coll. on Automata, Languages and Programming*, 2004.

[BC05] Mikolaj Bojanczyk and Thomas Colcombet. Tree-walking automata do not recognize all regular languages. In *Proc. ACM SIGACT Symp. on Theory of Computing*, 2005.

[CDG+99] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree Automata Techniques and Applications. Available at http://www.grappa.univ-lille3.fr/tata, 1999.

[EH05] Joost Engelfriet and Hendrik Jan Hoogeboom. Automata with Nested Pebbles Capture First-Order Logic with Transitive Closure. Technical Report 05-02, Leiden Institute of Advanced Computer Science, Leiden University, April 2005.

[EH99] Joost Engelfriet and Hendrik Jan Hoogeboom. Tree-walking pebble automata. In J. Karhumäki, H. Maurer, G. Paun, and G. Rozenberg, editors, *Jewels are forever, contributions to Theoretical Computer Science in honor of Arto Salomaa*, pages 72–83. Springer-Verlag, 1999.

[GH96] Noa Globerman and David Harel. Complexity results for two-way and multi-pebble automata and their logics. *Theoretical Computer Science*, 169(2):161–184, 1996.

[MSV03] Tova Milo, Dan Suciu, and Victor Vianu. Typechecking for XML transformers. *Journal of Computer and System Sciences*, 66(1):66–97, 2003.

[Sip78] Michael Sipser. Halting space-bounded computations. In *IEEE Conf. on Foundations of Computer Science*, pages 73–74, 1978.