Tree-Walking Pebble Automata *

Joost Engelfriet and Hendrik Jan Hoogeboom

Institute of Computer Science, Leiden University
P.O.Box 9512, 2300 RA Leiden, The Netherlands

Summary. The tree languages accepted by (finite state) tree-walking automata are known to form a subclass of the regular tree languages which is not known to be proper. They include all locally first-order definable tree languages. We allow the tree-walking automaton to use a finite number of pebbles, which have to be dropped and lifted in a nested fashion. The class of tree languages accepted by these tree-walking pebble automata contains all first-order definable tree languages and is still included in the class of regular tree languages. It also contains all deterministic top-down recognizable tree languages.

1 Introduction

One of the questions in tree language theory is whether a natural sequential automaton model exists for recognizing the regular tree languages. Of course, by definition, they are recognized by the bottom-up finite tree automaton. But that automaton is essentially parallel rather than sequential: the control of the automaton is at several nodes of the input tree simultaneously, rather than at just one. The top-down finite tree automaton is also parallel and, moreover, its deterministic version does not recognize all regular tree languages, which seems unnatural when compared to the case of strings. For strings, the (1-way, on-line) finite automaton was generalized to the 2-way (off-line) finite automaton, which also recognizes exactly the regular languages, see [16,15]. Here the point of view has changed: the input is not fed into the automaton (like money into a coffee machine), but the automaton walks on the input string (like a mouse in a maze). Clearly, this can be generalized to a sequential finite automaton on trees: the tree-walking automaton, introduced in [1]. The finite control of the tree-walking automaton is always at one node of the input tree. Based on the label of that node and on its child number (which is i if it is the i-th child of its parent, with $i = 0$ for the root), the automaton changes state and steps to one of the neighbouring nodes (parent or child). The tree-walking automaton of [1] was equipped with an underlying context-free grammar and with an output tape, to model syntax-directed translation from strings to strings. It did not

* Published in: "Jewels are forever, contributions to Theoretical Computer Science in honor of Arto Salomaa", J. Karhumäki, H. Maurer, G. Paun, G. Rozenberg (eds.), Springer-Verlag, 1999, pp. 72-83

need the test on the child number because that information could be coded into the nonterminals of the context-free grammar. As shown in [14], without the child number test the tree-walking automaton is not able to search the input tree in a systematic way, such as by a pre-order traversal. For this and other reasons the child number test is an essential feature of the tree-walking automaton.

Unfortunately, it is an open problem whether the tree-walking automaton recognizes all regular tree languages, and we conjecture that it does not, cf. [9]. Assuming the conjecture, there are two natural questions. How does the class TWA of tree languages accepted by tree-walking automata compare to other well-known subclasses of the regular tree languages? Which natural features can be added to the tree-walking automaton to obtain an automaton that *does* recognize the regular tree languages? To start with the second question, the main trouble with the tree-walking automaton seems to be that it gets lost rather easily: in a binary tree of which all internal nodes have the same label, all nodes look pretty much the same. One way of solving this is to extend the tree-walking automaton with a synchronized pushdown, for which pushing and popping is synchronized with moving down and up in the tree, respectively. It is shown (implicitly) in [10] and (explicitly) in [14] that this automaton recognizes exactly the regular tree languages. Another, classical remedy against getting lost is to use pebbles. For instance, arbitrary mazes can be searched by "maze-walking" finite automata with two pebbles, see [5]. The main aim of this paper is to investigate the power of tree-walking automata with pebbles. Obviously, the unrestricted use of pebbles leads to a class of tree languages much larger than the regular tree languages, in fact to all tree languages in NSPACE($\log n$). Thus, we restrict the automaton to the recursive use of pebbles, in the sense that the life times of pebbles, i.e., the times between dropping a pebble and lifting it again, are properly nested. A similar, but stronger, nesting requirement is studied in [13] for 2-way automata on strings. We prove in Section 5 that our restriction indeed guarantees that all tree languages recognized by the tree-walking pebble automaton are regular, but we conjecture that the automaton is not powerful enough to recognize *all* regular tree languages. In Section 6 we generalize the notion of pebble to that of a "set-pebble", in such a way that the tree-walking set-pebble automaton recognizes exactly the regular tree languages.

To answer the first question, we compare the TWA languages with the tree languages that can be defined in first-order logic, see [19] for a survey. One of the reasons that the regular tree languages are the natural generalization of the regular string languages to trees, is that they are precisely the tree languages definable in monadic second-order logic. This was, in fact, the main motivation for the introduction of finite tree automata in [7,18]. Thus, one way of investigating the power of several types of tree-walking automata is to compare them to several types of logics on trees. We show in Section 3 that TWA contains all tree languages that are definable in locally first-order

logic, where 'locally' means that the logic can talk about the parent-child relation between nodes of the tree, but not about the ancestor relation. We conjecture that it does not contain all first-order definable tree languages, but show in Section 5 that they can be recognized by the tree-walking pebble automaton. Note that TWA contains tree languages that are not first-order definable, such as the set of all trees with an even number of nodes (cf. [19]). As another answer to the first question, we conjecture in Section 4 that TWA does not contain all deterministic top-down recognizable tree languages, but show that they can be recognized by the tree-walking pebble automaton, with one pebble only.

2 Preliminaries

In this section we recall some well-known concepts and results concerning trees, logic for trees, and tree-walking automata. We use $\mathbb{N} = \{0, 1, 2, \ldots\}$, and for $m, n \in \mathbb{N}$, $[m, n] = \{i \mid m \leq i \leq n\}$.

Trees. For a ranked alphabet Σ, i.e., an alphabet Σ together with a function rank : $\Sigma \to \mathbb{N}$, the set of all trees over Σ is denoted T_Σ. A subset of T_Σ is called a tree language. As usual, a tree t over Σ is viewed as a finite, directed graph of which the nodes are labelled with symbols from Σ. By V_t we denote the set of nodes of t. Each node $v \in V_t$ has k children where k is the rank of the label of v. There is a linear order on these children which allows us to speak about the i-th child of v, and there is an edge with label i from v to its i-th child (for $1 \leq i \leq k$). The *child number* of a node v is i if it is the i-th child of its parent, and 0 if v is the root of t. For nodes u and v of t, $u \leq v$ denotes that u is an ancestor of v, i.e., that there is a directed (possibly empty) path from u to v. The yield of t is the string obtained by concatenating the labels of its leaves, from left to right. Finally, trees are denoted by terms in the usual way: $\sigma(t_1, \ldots, t_k)$ is the tree of which the root has label σ (of rank k), with direct subtrees $t_1, \ldots, t_k$.

Logic for Trees. We consider the same types of logic as in [19], viz. monadic second-order (MSO) logic, first-order (FO) logic, and locally first-order (LFO) logic. In [19] the latter two are called FO[<] logic and FO[S] logic, respectively.

For a ranked alphabet Σ, we consider monadic second-order formulas over Σ that describe properties of trees over Σ. This logical language has node variables $x, y, \ldots$, and node-set variables $X, Y, \ldots$. For a given tree t over Σ, node variables range over the elements of V_t, and node-set variables range over the subsets of V_t. There are five types of atomic formulas over Σ: $\mathrm{lab}_\sigma(x)$, for every $\sigma \in \Sigma$, meaning that x has label σ; $\mathrm{edg}_i(x, y)$, for every $i \leq$ the rank of a symbol in Σ, meaning that the i-th child of x is y; $x \leq y$, meaning that x is an ancestor of y; $x = y$, and $x \in X$, with obvious meaning. The formulas are built from the atomic formulas using the connectives $\neg$, $\wedge$, $\vee$, $\to$, and $\leftrightarrow$, as usual. Both node variables and node-set variables can

be quantified with $\exists$ and $\forall$. We will use $\mathrm{edg}(x,y)$ for the disjunction of all $\mathrm{edg}_i(x,y)$ (meaning that x is the parent of y), $\mathrm{root}(x)$ for $\neg\exists y(\mathrm{edg}(y,x))$, and $\mathrm{leaf}(x)$ for $\neg\exists y(\mathrm{edg}(x,y))$.

A *first-order* formula is an MSO formula that does not contain node-set variables, and a *locally first-order* formula is an FO formula that does not contain atomic formulas $x \leq y$.

For a closed formula ϕ over Σ and a tree $t \in T_\Sigma$, $t \models \phi$ denotes that t satisfies ϕ. If a formula ϕ has free variables, say, x, y, X, we also write $\phi(x,y,X)$. Moreover, $t \models \phi(u,v,U)$ denotes that t satisfies ϕ when x, y, X have value u, v, U, respectively (with $u, v \in V_t$ and $U \subseteq V_t$). The tree language defined by a closed formula ϕ over Σ is $L(\phi) = \{t \in T_\Sigma \mid t \models \phi\}$. $L(\phi)$ is called an MSO *definable tree language*. If ϕ is first-order, or locally first-order, then $L(\phi)$ is called an FO definable, or LFO definable tree language, respectively. According to the classical result of [7,18] (first shown for strings in [6,8]) the MSO definable tree languages are precisely the *regular tree languages*, i.e., the tree languages accepted by (bottom-up or top-down) finite tree automata. For more information on regular tree languages, see, e.g., [11,12].

We will also consider "trips" on trees (cf. [2,3,9]). A *trip*, or node relation, over Σ is a set of triples (t,u,v) where t is a tree over Σ, and $u, v \in V_t$. The trip defined by an MSO formula $\phi(x,y)$ with two free node variables x and y, is $T(\phi) = \{(t,u,v) \mid t \models \phi(u,v)\}$. $T(\phi)$ is called an MSO *definable trip*. In [9] the MSO definable trips are called the regular trips.

Tree-Walking Automata. A *tree-walking automaton*, or tw automaton, for short, is a (nondeterministic) finite state device that walks on a tree from node to node, following the edges of the tree (in either direction). At each moment of time the tw automaton is in a certain state, at a certain node of the input tree (over some ranked alphabet Σ). In one step, it can test the label and the child number of the current node, and move to the parent or to one of the children of the node, changing state. A child can be specified by its child number. The *language $L(A)$ accepted* by a tree-walking automaton A consists of all trees (over Σ) on which A has a computation that starts at the root of the input tree in its initial state, and ends in a final state. $L(A)$ will be called a *twa language*.

Since the tw automaton can test the child number of the current node (and hence, in particular, can test whether or not it is at the root), one of its basic capabilities is to make a pre-order traversal of the input tree (deterministically), starting and ending at the root: when entering a node v for the first time (from above), it moves up if v is a leaf, and otherwise moves down to v's first child; when entering v from below, it knows the number of the child u it just left (by testing u's child number and storing it in its finite control) and thus can move down to v's next child, or move up if u was the last child of v.

In [2,3] the tree-walking automaton is extended with logical capabilities: a *tree-walking automaton with* MSO *tests* is a finite state device as described

above, with the additional possibility of testing MSO properties of the current node. An MSO property is an MSO formula $\phi(x)$ with one free node variable; for the current node u of the input tree t, the automaton can test whether $t \models \phi(u)$. A given automaton can of course use only finitely many of these MSO tests. The language $L(A)$ accepted by such an automaton A is defined as above, but the main concern in [2,3] is with the *trip* $T(A)$ *computed* by A: the set of all (t, u, v) such that A can walk from u to v on t, starting in its initial state and ending in a final state. The main result of [2,3] is the following.

Proposition. *A trip is* MSO *definable if and only if it can be computed by a tree-walking automaton with* MSO *tests.*

It is an immediate consequence of (the if-direction of) this result that all tree languages accepted by tree-walking automata with MSO tests are MSO definable: if $T(A) = T(\phi(x, y))$, then $L(A) = L(\psi)$ where ψ is the formula $\exists x, y : \mathrm{root}(x) \wedge \phi(x, y)$. In particular, all twa languages are MSO definable and hence regular. However, as mentioned in the Introduction, it is an open problem whether every regular tree language is a twa language. Note that every regular tree language is a projection of a deterministic twa language. This is because every regular tree language is a projection of the set of derivation trees of a context-free grammar (cf. [12], Section 8), and every such derivation tree language can obviously be accepted by a deterministic tw automaton: the automaton traverses the input tree and checks for each node whether the labels of the node and its children form a production of the context-free grammar.

3 Locally First-Order Logic

As mentioned above, it is not known whether the tree-walking automaton can accept all MSO definable tree languages. Here we show that it can accept all locally first-order definable tree languages. The hard work for this has already been done: the proof is based on the characterization of the LFO definable tree languages as the so-called locally threshold testable tree languages ([19], Section 4.2). Intuitively, a tree language is locally threshold testable if membership of a tree in the language can be determined by looking at local spheres around all nodes of the tree, and count how many times these spheres occur, up to some threshold. To explain this formally we need some terminology.

Let Σ be a ranked alphabet. Consider a "radius" $r \in \mathbb{N}$. For a node u of a tree t over Σ, we denote by $\mathrm{sph}_r(t, u)$ the subgraph of t induced by all nodes of distance at most r to u (where distance is measured along undirected paths), with u as a designated node. Define $S_r = \{\mathrm{sph}_r(t, u) \mid t \in T_\Sigma, u \in V_t\}$. This is a finite set because we do not distinguish between isomorphic graphs. Now consider a "threshold" $q \in \mathbb{N}$. Define $F_{r,q}$ to be the (finite) set of all partial functions $f : S_r \to [0, q]$. For a tree t, define $f^t_{r,q} \in F_{r,q}$ such that for every

sphere $s \in S_r$, $f^t_{r,q}(s) = \#\{u \in V_t \mid \mathrm{sph}_r(t,u) = s\}$ provided this number is in $[0,q]$. Now, a tree language L over Σ is *locally threshold testable* if there exist $r, q \in \mathbb{N}$ and $F \subseteq F_{r,q}$ such that $L = \{t \in T_\Sigma \mid f^t_{r,q} \in F\}$.

It should be clear that every locally threshold testable tree language can be accepted by a deterministic tw automaton. The automaton A traverses the input tree t in pre-order and for each node u of t it computes the sphere $\mathrm{sph}_r(t,u)$ with centre u, in its finite control, by just searching systematically the neighbourhood of u within distance r. Thus, counting the number of occurrences of these spheres up to the threshold q, A can compute $f^t_{r,q}$ and check whether it is in F.

This shows that the deterministic tree-walking automaton can accept all LFO definable tree languages.

4 One Pebble

We conjecture that the twa languages form a proper subclass of the regular tree languages (cf. [9]) and propose the following tree languages L_1^{sp} and L_2^{sp} as counter-examples. Let $\Sigma = \{\sigma, a, b\}$ where σ has rank 2 and a, b have rank 0. Let L_1 be the set of all trees $t \in T_\Sigma$ such that all root-to-leaf paths of t have even length, and let L_2 be the set of all trees $t \in T_\Sigma$ such that the yield of t is in a^*ba^*. For $i = 1, 2$, the language L_i^{sp} consists of all trees $\sigma(t_1, \sigma(t_2, \ldots \sigma(t_k, \tau)\ldots))$ with $k \geq 0$, $\tau \in \{a, b\}$, and $t_1, \ldots, t_k \in L_i$. Thus, a tree t in L_i^{sp} consists of a *spine*, i.e., the path from the root of t to its rightmost leaf, such that the first child of each node on the spine is the root of a subtree which belongs to L_i. It should be clear that L_1^{sp} and L_2^{sp} are regular tree languages. We think that they cannot be accepted by a tw automaton.

Intuitively, the reason that no tree-walking automaton A can accept L_i^{sp} is the following. Let us say that a subtree of the input tree t is a *spine subtree* if its root is the first child of a node on the spine of t. Thus, A has to check that every spine subtree of t is in L_i. One way of doing this would be to move down node by node from the root along the spine, and for each node u on the spine check the spine subtree s of u. To do this, A has to visit all the leaves of s. But then A gets lost because it does not know when it has returned to the spine, i.e., to u. The only way for A to find out whether a node v is on the spine seems to be to move up as long as the child number is 2 and then test whether it is at the root, but then of course A has also got lost because it does not know how to return to v.

Note that it is not difficult to see that the *complement* of L_1^{sp} *can* be accepted by a tree-walking automaton: walk down nondeterministically to the root of one of the spine subtrees and then walk down nondeterministically to one of its leaves, checking that the path has odd length. Thus, if L_1^{sp} cannot be accepted by a tree-walking automaton, then the class of twa languages is not closed under complement. Moreover, it would imply that the deterministic tw automaton is less powerful than the nondeterministic one, because the class

of deterministic twa languages *is* closed under complement. The latter follows from the, maybe surprising, fact that every deterministic tw automaton can be simulated by one that is always halting (see [17]).

As mentioned in the Introduction, one way of not getting lost is to use pebbles. It should be clear that L_i^{sp} can be accepted by a (deterministic) tree-walking automaton with one pebble: the pebble is put on a spine node while the automaton is checking the corresponding spine subtree. A *one-pebble tree-walking automaton* (shortly, 1p-tw automaton) is a tree-walking automaton that additionally carries a pebble. It can drop the pebble on the current node, it can test whether or not the pebble is at the current node, and it can lift the pebble from the current node (when it lies there, of course). At the beginning and end of a computation the pebble should not be on the input tree. A 1p-tw automaton A accepts a tree language $L(A)$ in the usual way, and $L(A)$ is called a *1p-twa language*. It is well known that 2-way finite automata (on strings) with one pebble recognize the regular languages, see [4,13]. Similarly, it is shown in (Theorem 10 of) [9] that all 1p-twa languages are regular tree languages, see also the next section.

We now note that the language L_1^{sp} is, in fact, a deterministic top-down recognizable (dtr, for short) tree language, i.e., can be recognized by a deterministic top-down finite tree automaton (see, e.g., [11,12]). In fact, we will show that *every* dtr tree language can be accepted by a (deterministic) one-pebble tree-walking automaton. This is based on a well-known characterization of the dtr tree languages (see, e.g., [11], Theorem II.11.6) which we now recall. For a ranked alphabet Σ, we define the (non-ranked) *path alphabet* Π_Σ to consist of all symbols σ_i with $\sigma \in \Sigma$ and $1 \leq i \leq \mathrm{rank}(\sigma)$, plus all symbols of Σ of rank 0. Thus, for the ranked alphabet Σ of L_{sp}, $\Pi_\Sigma = \{\sigma_1, \sigma_2, a, b\}$. For a tree t, its *path language* $\mathrm{path}(t) \subseteq \Pi_\Sigma^*$ is defined recursively as follows: for τ of rank 0, $\mathrm{path}(\tau) = \{\tau\}$, and for σ of rank $k > 0$, $\mathrm{path}(\sigma(t_1, \ldots, t_k)) = \bigcup_{i=1}^{k} \sigma_i \cdot \mathrm{path}(t_i)$.

The characterization is: a tree language L over Σ is a dtr tree language if and only if there is a regular (string) language R over Π_Σ such that $L = \{t \in T_\Sigma \mid \mathrm{path}(t) \subseteq R\}$. As an example, the regular language R for L_{sp} is $\sigma_2^*(a \cup b \cup \sigma_1 R')$ where R' is the set of all strings of odd length.

Using this characterization, it is easy to see that all dtr tree languages are deterministic 1p-twa languages. Let R be a regular language over Π_Σ and let M be an ordinary deterministic finite automaton that accepts the mirror image of R. We describe a one-pebble tw automaton A that checks whether all paths of the input tree t are in R. The automaton A traverses t in preorder and for each leaf u of t it executes the following subroutine. It drops its pebble on u and walks up to the root, simulating M on the corresponding string in $\mathrm{path}(t)$. More precisely, it starts with the initial state of M and feeds M the label of u as input. For each node v on the path from u to the root, A determines the child number i of v, moves to the parent v' of v, and feeds M the symbol σ_i as input, where σ is the label of v'. At the root, A checks

that M has arrived in a final state. Finally, A returns to u by searching t for the pebble (traversing t deterministically), and lifts the pebble from u.

This shows that all deterministic top-down recognizable tree languages are accepted by deterministic tree-walking automata with one pebble.

We conjecture that the 1p-twa languages are still a proper subclass of the regular tree languages. The proposed counter-example is the language L_2^{nsp} which is obtained from $L = L_2^{\mathrm{sp}}$ by substituting $\rho(L)$ for each a and $\rho(L^{\mathrm{c}})$ for each b, where ρ is a symbol of rank 1 and L^{c} is the complement of L. Intuitively, to recognize a tree from L_2^{nsp} a 1p-tw automaton would need its pebble to recognize the outer occurrence of a tree from L and thus would have to behave like a tw automaton to recognize the nested occurrences of trees from L and L^{c}. Note that it is easy to accept L_2^{nsp} with a tree-walking automaton that uses *two* pebbles.

5 Nested Pebbles

Next we note that the language L_2^{sp} of Section 4 is first-order definable (but not locally). To define it, we use $x \leq_2 y$ to mean that $x \leq y$ and that all edges on the path from x to y have label 2. The latter property is expressible by the formula $\forall x', y' : (x \leq x' \wedge \mathrm{edg}(x', y') \wedge y' \leq y) \rightarrow \mathrm{edg}_2(x', y')$. Now $L_2^{\mathrm{sp}} = L(\phi)$ where ϕ is the formula $\forall x, y, z : (\mathrm{root}(x) \wedge x \leq_2 y \wedge \mathrm{edg}_1(y, z)) \rightarrow \psi(z)$, and $\psi(z)$ is $\exists! u : z \leq u \wedge \mathrm{leaf}(u) \wedge \mathrm{lab}_b(u)$. Similarly, it is easy to show that the language L_2^{nsp}, discussed at the end of Section 4, is first-order definable too.

As observed in Section 4, L_2^{sp} can be accepted by a tw automaton with a pebble, and L_2^{nsp} by one that uses two pebbles. Since we know from Section 3 that tw automata can recognize all locally first-order definable languages, this suggests that tw automata with pebbles can recognize all first-order definable languages. The key idea here is that a quantified variable corresponds to a pebble. However, as mentioned in the Introduction, the unrestricted use of pebbles leads out of the class of regular tree languages. Since quantifiers in a formula are properly nested, it seems natural to require that the life times of the pebbles are nested, where a life time of a pebble is the time between dropping it on a node and lifting it again (note that a pebble, like a cat, usually has more than one life). The obvious automaton for L_2^{nsp} indeed has nested pebble life times.

A *tree-walking automaton with nested pebbles* (shortly, p-tw automaton) is a tree-walking automaton that carries a finite number of pebbles, each with a unique name. In one step, it can determine which pebbles are lying on the current node, lift some of them, and drop some others. Initially and finally there should be no pebbles on the input tree. Moreover, the life times of the pebbles should be nested. This can be formalized by keeping a pushdown of pebble names in the configuration of the automaton, pushing a pebble when it is dropped and popping it when it is lifted. Note that since each pebble name occurs at most once in the pushdown, the automaton may keep track

of the pushdown in its finite control and thus "know" when it is allowed to drop or lift a pebble. A p-tw automaton A accepts a tree language $L(A)$ as usual, and $L(A)$ is called a *p-twa language*. We will also need the trip $T(A)$ computed by A (cf. Section 2): the set of all (t, u, v) such that A can walk from u to v on t, starting in its initial state and ending in a final state (with the above restrictions on pebbles).

In the remainder of this section we show that all first-order definable tree languages are deterministic p-twa languages, and that all p-twa languages are regular. The first proof is by induction on the structure of the formula. For each first-order formula ϕ we construct a deterministic always-halting p-tw automaton A that uses the (free or bound) variables of ϕ as pebble names. Without loss of generality we may assume that no variable occurs both free and bound in ϕ. For a given input tree t, the automaton A finds out whether or not ϕ is true for t. More precisely, let the free variables of ϕ be in the set $\{x_1, \ldots, x_k\}$, i.e., $\phi(x_1, \ldots, x_k)$. For nodes $u_1, \ldots, u_k$ of t, A receives as input the tree t with the pebbles $x_1, \ldots, x_k$ lying on the nodes $u_1, \ldots, u_k$, respectively, and starts its computation in its initial state at the root of t. During its computation it is not allowed to lift or drop the pebbles x_i (but they can of course be tested); the life times of the other pebbles should be nested. The computation of A should halt at the root of t with the pebbles $x_1, \ldots, x_k$ still lying on the nodes $u_1, \ldots, u_k$, and no other pebbles lying on t. Finally, A halts in a final state if and only if $t \models \phi(u_1, \ldots, u_k)$. The construction of A is easy for the atomic formulas. As an example, the automaton for $x \leq y$ searches the input tree for the node u with pebble x, traverses the subtree with root u to see whether it contains the pebble y, and returns to the root. In the induction step it suffices to consider negation, conjunction, and universal quantification. For negation, just interchange final and non-final states (note that the automaton is deterministic and always-halting). For conjunction, just simulate the two given automata, one after the other. Now consider a formula $\phi = \forall x : \phi'(x)$ and let A' be the automaton constructed for ϕ'. The automaton A for ϕ traverses the input tree in pre-order and executes the following subroutine for every node u: it drops pebble x on u, walks to the root, simulates A', returns to pebble x, and lifts pebble x from u.

This proves that all FO definable tree languages can be accepted by a deterministic tree-walking automaton with nested pebbles. Note that the automaton can be constructed in such a way that it uses k pebbles where k is the quantifier nesting depth of the formula.

Next we prove that all p-twa languages are regular. Let A be a p-tw automaton over the ranked alphabet Σ. We will show, by induction on the number of pebbles, that the trip $T(A)$ can be computed by a tree-walking automaton M with MSO tests, i.e., $T(M) = T(A)$. By the Proposition in Section 2 and the argument following it, this implies that all p-twa languages are MSO definable and hence regular. The basic idea is that a subcomputation

of A that corresponds to the life time of a pebble on node u, can be replaced by an MSO test on u. Without loss of generality we assume that the pebble names of A are $1, \ldots, n$, and that at each moment of time pebbles $1, \ldots, i$ are lying on the input tree, for some $0 \leq i \leq n$. The induction is on n. For $n = 0$, A is an ordinary tw automaton (and so we take $M = A$). Assume now that the result holds for $n-1$. Consider a subcomputation of A that corresponds to a life time of pebble 1. Obviously, during this life time, A behaves like a p-tw automaton with $n-1$ pebbles on an input tree of which one node is "marked". Let Σ' be the ranked alphabet $\Sigma \cup \{\sigma' \mid \sigma \in \Sigma\}$, where each σ' is a new ("marked") symbol, of the same rank as σ. Define A' to be the tw automaton with pebbles $\{2, \ldots, n\}$ that behaves in the same way as A, interpreting a node with label σ' as a node with label σ that contains pebble 1; A' has initial state p and final state q, where p and q are the states of A at the start and end of the considered life time, respectively. By the induction hypothesis and the Proposition in Section 2, $T(A')$ is MSO definable, i.e., there is a formula $\phi'(x_1, x_2)$ over Σ' such that $T(A') = T(\phi')$. Let $\phi(x_1, x_2, y)$ be the formula that is obtained from $\phi'(x_1, x_2)$ by changing every atomic subformula $\mathrm{lab}_{\sigma'}(z)$ into the formula $\mathrm{lab}_{\sigma}(z) \wedge z = y$ and every atomic subformula $\mathrm{lab}_{\sigma}(z)$ into $\mathrm{lab}_{\sigma}(z) \wedge z \neq y$. Then, for all nodes u_1, u_2, v of $t \in T_\Sigma$, $t \models \phi(u_1, u_2, v)$ iff A can walk from u_1 in state p to u_2 in state q with pebble 1 on v (without lifting pebble 1). Consequently, the considered subcomputation can be replaced by the MSO test $\phi(x, x, x)$ on the current node. Thus, A is turned into a tw automaton with MSO tests that computes the same trip. For more details see Chapter 2 of [20] where this result is shown for the special case of 2-way automata on strings.

This proves that all tree languages accepted by tree-walking automata with nested pebbles are regular. We conjecture that they form a proper subclass of the regular tree languages, and that the number of pebbles determines a proper hierarchy (by iterating the construction at the end of Section 4).

6 Set-Pebbles

It was shown in the previous section that the FO definable tree languages can be recognized by tree-walking automata that use their pebbles to implement the quantification of node variables. Thus, one naturally gets the idea that all MSO definable tree languages (i.e., all regular tree languages) can be recognized by an automaton that uses a generalized type of pebble by which the quantification of node-set variables can be implemented. We will call such a generalized pebble a "set-pebble". Let us define a (nondeterministic) *tree-walking automaton with set-pebbles* to be a tree-walking automaton that carries a finite number of set-pebbles (with distinct colours) that it can drop, test, and lift. Dropping a set-pebble of a certain colour means that, nondeterministically, a pebble of that colour is dropped on any number of nodes of the input tree. Lifting a set-pebble of a certain colour means that

all pebbles of that colour are lifted from the tree. The automaton can test the colours of the pebbles that lie on the current node. Note that dropping and lifting are independent of the current node. Finally, as for pebbles, we require the life times of the set-pebbles to be nested.

Let us now argue that the tree-walking automaton with set-pebbles accepts exactly the regular tree languages. In one direction, it is well known (see [19]) that every regular tree language is MSO definable by a formula of the form $\phi = \exists X_1, \ldots, X_k : \phi'(X_1, \ldots, X_k)$ where ϕ' is locally first-order (in the sense that it does not contain $\exists X$, $\forall X$, and $x \leq y$). Since, by Section 3, LFO formulas can be accepted by tw automata, it should be clear that, to implement ϕ, it suffices to use set-pebbles with colours $X_1, \ldots, X_k$, dropping them at the start of the computation, then simulating the tw automaton that implements ϕ', and lifting them at the end of the computation. In the other direction we follow the proof in Section 5 that all p-twa languages are regular. As in that proof, consider a subcomputation of A that corresponds to a life time of the set-pebble with colour 1. Suppose that A drops the set-pebble in state p when it is at node u, and lifts it again in state q when it is at node v. This means that there is a set of nodes U (the nodes that are covered by the set-pebble) such that A walks from u to v behaving like a tw automaton with $n-1$ set-pebbles on an input tree t of which the nodes in U are "marked". Thus, by the induction hypothesis and the Proposition in Section 2, there is a formula $\phi(x_1, x_2, X)$ that models this behaviour (for $x_1 = u$, $x_2 = v$, and $X = U$). Hence t satisfies $\psi(u, v)$, where ψ is the formula $\exists X : \phi(x_1, x_2, X)$. In other words, (t, u, v) is in the trip $T(\psi)$. By the Proposition in Section 2 there is a tw automaton M with MSO tests such that $T(M) = T(\psi)$. Consequently, the considered subcomputation of A can be replaced by a simulation of M. This turns A into a tw automaton with MSO tests.

We finally note that is not clear how one could define a deterministic version of the tree-walking automaton with set-pebbles.

Acknowledgment. We are grateful to Jan-Pascal van Best for helping us with the proof of the second result of Section 5.

References

1. A.V. Aho and J.D. Ullman; Translations on a context free grammar, Inform. and Control 19 (1971), 439–475
2. R. Bloem, J. Engelfriet; Monadic second order logic and node relations on graphs and trees, in J. Mycielski, G. Rozenberg, and A. Salomaa, editors, *Structures in Logic and Computer Science*, Lecture Notes in Computer Science 1261, Springer-Verlag, 1997, pp.144-161
3. R. Bloem, J. Engelfriet; Characterization of properties and relations defined in monadic second order logic on the nodes of trees, Tech. Report 97–03, Leiden University, August 1997 `http://www.wi.LeidenUniv.nl/TechRep/1997/tr97-03.html`

4. M. Blum, C. Hewitt; Automata on a 2-dimensional tape, in Proc. 8th IEEE Symp. on Switching and Automata Theory, pp.155–160, 1967
5. M. Blum, D. Kozen; On the power of the compass (or, why mazes are easier to search than graphs), Proc. 19th FOCS (Annual Symposium on Foundations of Computer Science), 1978, pp.132–142
6. J. Büchi; Weak second-order arithmetic and finite automata, Z. Math. Logik Grundlag. Math. 6 (1960), 66–92
7. J. Doner; Tree acceptors and some of their applications, J. of Comp. Syst. Sci. 4 (1970), 406–451
8. C.C. Elgot; Decision problems of finite automata and related arithmetics, Trans. Amer. Math. Soc. 98 (1961), 21–51
9. J. Engelfriet, H.J. Hoogeboom, J.P. van Best; Trips on trees, Manuscript, 1999, to appear in Acta Cybernetica
10. J. Engelfriet, G. Rozenberg, G. Slutzki; Tree transducers, L systems, and two-way machines, J. of Comp. Syst. Sci. 20 (1980), 150-202
11. F. Gécseg, M. Steinby; *Tree Automata*, Akadémiai Kiadó, Budapest, 1984
12. F. Gécseg, M. Steinby; Tree Languages, in G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, *Volume 3: Beyond Words*, Chapter 1, Springer-Verlag, 1997
13. N. Globerman, D. Harel; Complexity results for two-way and multi-pebble automata and their logics, Theor. Comput. Sci. 169 (1996), 161–184
14. T. Kamimura, G. Slutzki; Parallel and two-way automata on directed ordered acyclic graphs, Inf. and Control 49 (1981), 10–51
15. M.O. Rabin, D. Scott; Finite automata and their decision problems, IBM J. Res. Devel. 3 (1959), 115–125
16. J.C. Shepherdson; The reduction of two-way automata to one-way automata, IBM J. Res. Devel. 3 (1959), 198–200
17. M. Sipser; Halting space-bounded computations, Proc. 19th FOCS (Annual Symposium on Foundations of Computer Science), 1978, pp.73–74
18. J.W. Thatcher, J.B. Wright; Generalized finite automata theory with an application to a decision problem of second-order logic, Math. Systems Theory 2 (1968), 57–81
19. W. Thomas; Languages, Automata, and Logic, in G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, *Volume 3: Beyond Words*, Chapter 7, Springer-Verlag, 1997
20. J.P. van Best; *Tree–Walking Automata and Monadic Second Order Logic*, Master's Thesis, Leiden University, July 1998 `http://www.wi.LeidenUniv.nl/MScThesis/IR98-06.html`