On the Power of Tree-Walking Automata

Frank Neven[⋆]
Limburgs Universitair Centrum and Thomas Schwentick
Johannes Gutenberg-Universität Mainz
Institut für Informatik

No Institute Given

Abstract. Tree-walking automata (TWAs) recently received new attention in the fields of formal languages and databases. Towards a better understanding of their expressiveness, we characterize them in terms of transitive closure logic formulas in normal form. It is conjectured by Engelfriet and Hoogeboom that TWAs cannot define all regular tree languages, or equivalently, all of monadic second-order logic. We proof this conjecture for a restricted, but powerful, class of TWAs. In particular, we show that 1-bounded TWAs, that is TWAs that are only allowed to traverse every edge of the input tree at most once in every direction, cannot define all regular languages. We then extend this result to a class of TWAs that can simulate first-order logic (FO) and is capable of expressing properties not definable in FO extended with regular path expressions; the latter logic being a valid abstraction of current query languages for XML and semi-structured data.

[⋆] Research Assistant of the Fund for Scientific Research, Flanders.

1 Introduction

Regular tree languages can be defined by means of many equivalent formalisms, for instance: (non)deterministic bottom-up and nondeterministic top-down tree automata, alternating tree automata, two-way tree automata, homomorphic images of local tree languages, and monadic second-order logic [GS97,Tho97a]. However, it is not known whether there exists a natural inherently sequential model for recognizing the regular tree languages. Of course, by definition, they are recognized by bottom-up finite tree automata, but these automata are essentially parallel rather than sequential: the control of the automata is at several nodes of the input tree simultaneously, rather than at just one. With this aim in mind, Engelfriet, together with his co-workers Bloem, Hoogeboom, and van Best, initiated a research program [BE,EH99a,EHvB99] studying (extensions of) the tree-walking automata (TWAs) originally introduced by Aho and Ullman [AU71]. The finite control of a tree-walking automaton is always at one node of the input tree. Based on the label of that node and its child number (which is i if it is the ith child of its parent), the automaton changes state and steps to one of the neighboring nodes (parent or child). Without the test on the child number such automata cannot even search the tree in a systematic way, such as by a pre-order traversal as is shown by Kamimura and Slutzki [KS81]. However, also with the child test, it is conjectured that these automata cannot express all regular tree languages [EH99a,EHvB99]. In this paper, we study the expressiveness of tree-walking automata by characterizing them in terms of transitive closure logic formulas in normal form and prove the above mentioned conjecture for a restricted, but powerful, class of tree-walking automata.

Apart from the above purely theoretical motivation, recently, new interest in tree-walking automata emerged from the field of database theory. Indeed, one of the major research topics at the moment is the design and study of query languages for the manipulation of XML documents or electronic documents in general [ABS99,N99]. Such documents are usually modeled by ordered labeled trees or graphs, depending on the application at hand. In this research, tree-walking automata are used for various purposes and appeared in various forms. Milo, Suciu, and Vianu [MSV], for instance, used a transducer model based on tree-walking automata as a formal model for an XML transformer encompassing most current XML transformation languages. Brüggeman-Kleinn, Hermann, and Wood [BHW99], proposed to use *caterpillar expressions* as a pattern language for XML transformation languages. Interestingly, caterpillar expressions relate to tree-walking automata like regular expressions relate to string automata: they are just a different, though a lot more user friendly, representation of the same thing. Furthermore, they conjectured their formalims to be less expressive than the regular tree languages. Another, more direct, occurrence of tree-walking automata is embodied in the actual XML transformation language XSLT [Cla99] proposed by the World Wide Web consortium (W3C) and currently being implemented by IBM. In formal language theoretic terms, this query language can be best described as a *tree-walking tree transducer* [BMN99]. Hence, results on the expressiveness of tree-walking automata could give insight in the expressiveness of actual XML transformation languages.

We start by characterizing the expressiveness of (deterministic and nondeterministic) tree-walking automata in terms of *(deterministic and non-deterministic) transitive closure logic* (DTC and TC) formulas in normal form. That is, formulas of the form $[(\mathrm{D})\mathrm{TC}(\varphi)](\varepsilon, \varepsilon)$, where φ is an FO formula containing predicates $\mathrm{depth}_m(x)$ defining x as a vertex whose depth is a multiple of m; and where ε refers to the root of the tree under consideration. Our result thus implies that any lower bound on (D)TC formulas in normal form is also a lower bound for (non)deterministic tree-walking automata. Unfortunately, proving lower bounds for the latter logic does not seem much easier than the original problem as Ehrenfeucht games for DTC and TC are quite involved. Therefore, we use a direct approach for a restricted, but expressive, class of tree-walking automata in the hope that these techniques will provide insight for the general case.

We first show that 1-bounded tree-walking automata, that is tree-walking automata that are only allowed to traverse every edge of the input tree at most once in every direction, cannot define all regular languages. In particular, we obtain that they can not evaluate tree-structured Majority circuits where the gates have fan-in greater than 2. Next, we generalize this result to a rather powerful class of tree-walking automata, called r-restricted. These automata are rather expressive as they can define all of first-order logic (FO) and are capable of expressing some tree languages not definable in FO extended with regular path expressions. The latter logic is an abstraction of current query languages for semi-structured data and XML [ABS99,N99], and, for instance, cannot define the set of trees representing Boolean circuits evaluating to true [NS] which can easily be defined by r-restricted tree-walking automata.

We conclude by mentioning some related work. Bargury and Makowsky [MSV] proved an equivalence between transitive closure logic and two-way multihead automata operating on grids. Their simulation of automata involves nesting of TC operators. Potthoff [Pot94] showed that the same normal form of TC we use, suffices to define all regular *string* languages, the opposite direction being trivial in the string case. Recently, Engelfriet and Hoogeboom [EH99b] showed that tree-walking automata with pebbles correspond exactly to TC. Hence, when allowing pebbles one can simulate nested TC operators. Fülöp and Maneth [FM] recently showed that the domains of partial attributed tree transducers correspond to the tree-walking automata in universal acceptance mode.

This article is structured as follows. In Section 2, we define tree-walking automata. In Section 3, we proof the logical characterization of tree-walking automata in terms of transitive closure logic, and in Section 4 we proof the Engelfriet and Hoogeboom conjecture for two restrictions of tree-walking automata.

2 Preliminaries

Trees. A *tree domain* τ *over* $\mathbb{N}$ is a subset of $\mathbb{N}^*$, such that if $v \cdot i \in \tau$, where $v \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $v \in \tau$. Here, $\mathbb{N}$ denotes the set of natural numbers. If $i > 1$ then also $v \cdot (i-1) \in \tau$. The empty sequence, denoted by ε, represents the root. We call the elements of τ *vertices.* A vertex w is a *child* of a vertex v (and v the *parent* of w) if $vi = w$, for some i. A Σ*-tree* is a pair $t = (\mathrm{dom}(t), \mathrm{lab}_t)$, where $\mathrm{dom}(t)$ is a tree domain over $\mathbb{N}$, and lab_t is a function from $\mathrm{dom}(t)$ to Σ. The *arity* of a tree is the maximum number of children of its vertices. We only consider trees with a fixed arity. The depth of a vertex u, denoted by $\mathrm{depth}(u)$ is the length of u (interpreted as a string over $\mathbb{N}^*$). The *distance* between two vertices u and v, denoted by $d(u,v)$ is defined as the length of the path between u and v where we assume that $d(u,u) = 0$.

A Σ-tree $\mathbf{t}$ can be naturally viewed as a finite structure (in the sense of mathematical logic [EF95]) over the binary relation symbols E and $<$, and the unary relation symbols $(O_\sigma)_{\sigma \in \Sigma}$. E is the edge relation and equals the set of pairs $(v, v \cdot i)$ for every $v, v \cdot i \in \mathrm{dom}(\mathbf{t})$. The relation $<$ specifies the ordering of the children of a node, and equals the set of pairs $(v \cdot i, v \cdot j)$, where $i < j$ and $v \cdot j \in \mathrm{dom}(\mathbf{t})$. For each σ, O_σ is the set of nodes that are labeled with a σ.

Tree-Walking Automata. Tree-walking automata (TWAs) can be seen as the simplest analogon of two-way string automata. A TWA starts its computation in an initial state at the root of the input tree. In each step, it moves to a neighbour vertex of the current vertex (or stays at the current vertex) and enters a state. The direction of movement and the new state depend only on the current state, the symbol at the current vertex and the child number of the current vertex, i.e., the relative position of the current vertex in the ordered list of the children of its parent.

More formally, a (k-ary) TWA is a tuple $(S, \Sigma, \delta, s_0, F)$, where S is the set of states, Σ is an alphabet (the set of possible vertex labels), $s_0 \in S$ is the initial state and $F \subseteq S$ is the set of accepting states. The only part where a TWA is formally different from a standard string automaton is the transition function δ. The transition function δ of a TWA is the union of the functions δ^i and $\delta^{\mathrm{root},i}$, where $i \in \{0, \ldots, k\}$. For a deterministic TWA,

- $\delta^{\mathrm{root},i}$ is a function from $S \times \Sigma$ to $\{\mathrm{stay}, \downarrow_1, \ldots, \downarrow_i\} \times S$, and
- for each $i \in \{0, \ldots, k\}$, δ^i is a function from $\{1, \ldots, k\} \times S \times \Sigma$ to $\{\uparrow, \mathrm{stay}, \downarrow_1, \ldots, \downarrow_i\} \times S$.

For a nondeterministic TWA the ranges of these functions are the respective power sets. If several TWAs are around we write δ_M to denote the transition function of TWA M and the like.

A configuration $c = [v, s]$ of a TWA M on a tree t consists of a vertex v of t and a state s of M. The immediate successor configuration c' of a configuration $[v, s]$ is defined as follows.

- If $v = \epsilon$, v has i children and carries the symbol σ then
  - $c' = [\epsilon, s']$, if $\delta^{\text{root},i}(s, \sigma) = (\text{stay}, s')$, and
  - $c' = [j, s']$, if $\delta^{\text{root},i}(s, \sigma) = (\downarrow_j, s')$.
- If $v = wj$, for some $j \leq k$, v has i children and carries the symbol σ then
  - $c' = [w, s']$, if $\delta^i(j, s, \sigma) = (\uparrow, s')$,
  - $c' = [v, s']$, if $\delta^i(j, s, \sigma) = (\text{stay}, s')$, and
  - $c' = [vj', s']$, if $\delta^i(j, s, \sigma) = (\downarrow_{j'}, s')$.

We write $c \Rightarrow_{M,t} c'$ to express that c' is the immediate successor configuration of c in the computation of M on t. Following standard convention we write $c \Rightarrow^j_{M,t} c'$ ($c \Rightarrow^*_{M,t} c'$) to express that c' is the j-th (some) successor configuration of c in the computation of M on t. If M and/or t are clear from the context we may omit them.

Example 1. We illustrate the above definition by means of an example. In particular we define a deterministic tree-walking automaton that accepts all tree-structured Boolean circuits of fan-in 2 that evaluate to true. A similar construction can be given for each fixed bound on the fan-in. For convenience, we only consider circuits of the right format. That is, all inner nodes and the root have exactly two children and are labeled with AND or OR. Further, all leaves are labeled by 0 or 1. These circuits are assigned a truth value in the usual way. Define M as the tuple $(S, \Sigma, \delta, \text{eval}, F)$ with $S = \{\text{eval}, 0, 1, \text{left-child-0}, \text{left-child-1}\}$, $\Sigma = \{\text{AND}, \text{OR}, 0, 1\}$, and $F = \{1\}$. The transition function δ is defined as follows:

- M starts by evaluating the first subtree of the root:
  for all $\sigma \in \{\text{AND}, \text{OR}\}$, $\delta^{\text{root},2}(\text{eval}, \sigma) = (\downarrow_1, \text{eval})$;
- When reaching a zero (one) leaf, which is a left child, M moves up with this information:

$$\begin{aligned} \delta^0(1, \text{eval}, 0) &= (\uparrow, \text{left-child-0}); \\ \delta^0(1, \text{eval}, 1) &= (\uparrow, \text{left-child-1}); \end{aligned}$$

- If M enters an inner vertex from a left child it is always in one of the two states left-child-0 or left-child-1. If the current vertex is an AND vertex and the state is left-child-0 then the current vertex will evaluate to 0 no matter the right subtree. An analogous statement holds for OR-gates and the state left-child-1. The information passed to the parent vertex depends on whether the current vertex is itself a left or a right child. If the outcome of the left child is not sufficient to determine the value of the current vertex M has to enter its right child. Formally, for each $i \in \{1, 2\}$:

$$\begin{aligned} \delta^2(1, \text{left-child-0}, \text{AND}) &= (\uparrow, \text{left-child-0}); \\ \delta^2(2, \text{left-child-0}, \text{AND}) &= (\uparrow, 0); \\ \delta^2(1, \text{left-child-1}, \text{OR}) &= (\uparrow, \text{left-child-1}); \\ \delta^2(2, \text{left-child-1}, \text{OR}) &= (\uparrow, 1); \\ \delta^2(i, \text{left-child-1}, \text{AND}) &= (\downarrow_2, \text{eval}); \text{ and} \\ \delta^2(i, \text{left-child-0}, \text{OR}) &= (\downarrow_2, \text{eval}). \end{aligned}$$

- If M enters an inner vertex from a right child it is always in one of the states 0 or 1. By what we have said before, this state indicates the value of the subtree at the current vertex. Hence, it only has to be passed to its parent vertex. Consequently, for each $i \in \{0, 1\}$ and $\sigma \in \{\text{AND}, \text{OR}\}$:

$$\begin{aligned} \delta^2(1, 0, \sigma) &= (\uparrow, \text{left-child-0}); \\ \delta^2(1, 1, \sigma) &= (\uparrow, \text{left-child-1}); \text{ and} \\ \delta^2(2, i, \sigma) &= (\uparrow, i). \end{aligned}$$

– It remains to handle the case of leaves that are right children of their parent. They simply have to pass their value to the parent. For each $i \in \{0, 1\}$:

$$\delta^0(2, \text{eval}, i) = (\uparrow, i).$$

It will be convenient to subsume the overall effect of a TWA on a subtree of a tree in a so-called behaviour function. Intuitively, if f is the behaviour function of a subtree t then $f(s) = s'$ if and only if in the computation of M which starts at the root v of t in state s the parent of v is entered in state s'. Obviously, the behaviour function of a subtree t depends on the child number of its root in the full tree.

We define behaviour functions more formally. Let M be a k-ary deterministic TWA and let t be a (at most) k-ary tree. For $i \leq k$, let $t(i)$ denote the tree which consists of a root which has the root of t as the i-th child and $i - 1$ other children which are leaves. The behaviour function $f_{M,t,i}$ of M on t as an i-th child maps states of M to states of M. It is defined as follows. If s is a state of M then $f_{M,t,i}(s) = s'$, if there is a j such that $(i, s) \Rightarrow^j_{M,t(i)} (\epsilon, s')$ and j is minimal with this property.

Note that $f_{M,t,i}$ does not depend on the labels of the vertices outside of t. Furthermore it does not depend either on the actual embedding of t in a larger tree as long as the root of t is an i-th child.

For non-deterministic TWAs behaviour functions are defined analogously but with sets of states as function values.

Next, we turn to the definition of some classes of restricted TWAs.

– We call a TWA 1-*bounded*, if, for all trees t, it traverses each edge of t at most once in each direction.
– We call a TWA M r-*restricted* if the following holds. For each pair u, v of vertices of a tree t such that $d(u, v) > r$ the computation of M on t does not contain 4 configurations $[u, s_1], [v, s_2], [u, s_3], [v, s_4]$ in the given order. Intuitively, this means that each path of length more than r is traversed at most once in each direction.

Clearly, each 1-bounded TWA is also r-restricted for every $r \geq 1$. In Proposition 1, we show that the latter automata can define all of first-order logic (FO). Moreover, r-restricted TWAs can even define some tree languages not definable in FO extended with regular path expressions. The latter logic is an abstraction of current query languages for semi-structured data and XML [ABS99], and, for instance, cannot define the set of trees representing Boolean circuits evaluating to true [NS]. In Section 4, we show that r-restricted TWAs cannot define the set of all regular tree languages thereby giving an answer to the conjecture of Engelfriet and Hoogeboom for a powerful class of TWAs.

Let, for $m > 0$, depth_m be a unary relation symbol. In the following we will consider trees which have additionally the predicate depth_m, for some m. In all trees, depth_m will contain all vertices the depth of which is a multiple of m. For a vertex u of a tree t, its r-sphere $S^t_r(u)$ is the set $\{v \mid d(u, v) \leq r\}$. For a tuple of vertices $\bar{u}$, define $S^t_r(\bar{u})$ as $\bigcup_{u \in \bar{u}} S^t_r(u)$. We define its r-neighborhood $N^t_r(\bar{u})$ as the structure t extended with the constants $\bar{u}$ restricted to the set $S^t_r(\bar{u})$.

3 A logical characterization of tree-walking automata

We characterize tree-walking automata by *transitive closure logic formulas* of the form $\text{TC}[\varphi(x, y)](\varepsilon, \varepsilon)$, where φ is an FO formula, which may make use of the predicate depth_m, for some m. We say that such a formula is in *normal form*. Further,

$$t \models \text{TC}[\varphi(x, y)](\varepsilon, \varepsilon)$$

iff the pair $(\varepsilon, \varepsilon)$ is in the transitive closure of the relation $\{(u,v) \mid t \models \varphi[u,v]\}$. We use *deterministic transitive closure logic formulas* (DTC) in an analogously defined normal form to capture deterministic tree-walking automata. In particular,

$$t \models \mathrm{DTC}[\varphi(x,y)](\varepsilon,\varepsilon)$$

iff the pair $(\varepsilon, \varepsilon)$ is in the transitive closure of the relation $\{(u,v) \mid t \models \varphi[u,v] \wedge (\forall z)(\varphi[u,z] \rightarrow z = v)\}$. The latter expresses that we disregard vertices u that have multiple φ-successors.

Before we prove the characterization, we show first, as an appetizer, how TWAs can evaluate FO sentences. The TWAs used for this task are of the restricted type introduced in Section 2.

Proposition 1. *Let φ be an FO sentence which can make use of the predicate depth_m, for some m. There exists an r and an r-restricted TWA accepting exactly the class of trees defined by φ.*

Proof. Our proof is an easy application of Hanf's Theorem (see, e.g., [EF95]). The same result for FO sentences without modulo depth predicates is obtained by Engelfriet and Hoogeboom [EH99a]. Consider a "threshold" q and a radius r. For a tree t, define the function $f^t_{r,q}$ mapping each isomorphism type τ of r-neighborhoods with one distinguished vertex to a number in the interval $[0,q]$, as follows,

$$f^t_{r,q}(\tau) := \begin{cases} |\{u \in \mathrm{dom}(t) \mid N^t_r(u) \cong \tau\}| & |\{u \in \mathrm{dom}(t) \mid N^t_r(u) \cong \tau\}| \leq q, \\ q & \text{otherwise.} \end{cases}$$

By Hanf's Theorem there are r and q such that, for every tree t, whether $t \models \varphi$ only depends on $f^t_{r,q}$. The latter function can readily be computed by a $2r$-restricted TWA. Indeed, the automaton traverses the input tree in pre-order and for each vertex u of t it computes the substructure $N^t_r(u)$ in its finite control by searching systematically the neighborhood of u within distance r and keeping track of the depth modulo m of the current vertex. The latter can be done in a straightforward way by keeping a counter in the state which is increased (modulo m) when the automaton moves down and decreased (modulo m) when the automaton moves up. □

The proof of the next lemma is an easy extension of a proof of Potthoff [Pot94] who characterized two-way *string* automata by means of TC formulas in normal form.

Lemma 1. *Every deterministic tree-walking automaton is definable by a DTC formula in normal form. Every nondeterministic tree-walking automaton is definable by a TC formula in normal form.*

Proof. We start with nondeterministic tree-walking automata. Therefore, let M be a nondeterministic tree-walking automaton with state set $\{0, \ldots, m-1\}$ and initial state 1. W.l.o.g., we can assume that if M accepts a tree it does so at its root in state 0.

We start with some notation. For a vertex u, we denote by $\mathrm{anc}_m(u)$ the closest ancestor of u (including u) whose depth is a multiple of m. We denote by $\mathrm{desc}_m(u)$ the set of closest descendants of u (u *not* included) whose depths are a multiple of m. By $t^{\downarrow m}_u$ we denote the subtree t_u where we delete all the subtrees rooted at vertices in $\mathrm{desc}_m(u)$.

We only keep track of the states M assumes at vertices occurring on depths that are multiples of m. We call such vertices (and configurations with such vertices) *important*. We associate each important configuration $[v,i]$, with the i-th vertex $v(i)$ of $t^{\downarrow m}_v$ in the DFS-order. We are going to construct a FO-formula φ such that the following holds. If v, w are two important vertices then M reaches the configuration $[w,j]$ from $[v,i]$ without passing any other important configuration, if and only if[1] $t \models \varphi(v(i), w(j))$.

[1] We make an exception from that rule for $v = \epsilon$. it may hold $t \models \varphi(\epsilon, w(j))$ whenever $[w,j]$ can be reached from $[\epsilon, 1]$ without passing important vertices.

As the computation between two successive important configurations only involves a subtree of bounded size it is clear that such a formula φ can be defined.

Then $t \models \mathrm{TC}[\varphi(x,y)](\varepsilon,\varepsilon)$ iff M accepts t.
When M is deterministic, then $t \models \mathrm{DTC}[\varphi(x,y)](\varepsilon,\varepsilon)$ iff M accepts t. $\square$

Theorem 1. *Nondeterministic tree-walking automata accept precisely the tree languages definable by TC formulas in normal form. Deterministic tree-walking automata accept precisely the tree languages definable by DTC formulas in normal form.*

Proof. We first restrict attention to nondeterministic automata. By Lemma 1, it suffices to show that the set of tree languages definable by TC formulas in normal form can be recognized by nondeterministic tree-walking automata. Therefore, let $\mathrm{TC}[\varphi(x,y)](\varepsilon,\varepsilon)$ be such a formula.

By Gaifman's Theorem (see, e.g., [Ga82]), there exists an r such that φ is equivalent to a Boolean combination of sentences χ and r-local formulas $\xi(x,y)$. Here, a formula $\xi(x,y)$ is r-local if for every tree t with vertices u and v, whether $t \models \xi[u,v]$ only depends on the isomorphism type of $N_r^t(u,v)$. As the rank of trees is fixed, there are only finitely many possible isomorphism types. M can evaluate the sentences χ at the begin of the computation as it is described in the proof of Proposition 1. Therefore, if $d(u,v) \leq 2r$ then M can check whether $t \models \varphi[u,v]$ by inspecting $N_{3r}^t(u)$, otherwise by first inspecting $N_r^t(u)$ and afterwards $N_r^t(v)$.

Now suppose the automaton arrives at a vertex u (with $u = \varepsilon$ as the first case). First, the automaton nondeterministically, whether it will go to a vertex v of distance $\leq 2r$ or to a vertex v of distance $> 2r$. In the first case, it inspects $N_{3r}^t(u)$ and chooses a v such that $t \models \varphi[u,v]$ if this is possible. Otherwise it first computes the type of $N_r^t(u)$, moves to a vertex v of distance $> 2r$ and checks that the type of $N_r^t(v)$ implies that $t \models \varphi[u,v]$.

Finally, if v is the root, the automaton accepts. Otherwise, it proceeds in the same manner. Clearly, the automaton will eventually accept if $\mathrm{TC}[\varphi(x,y)](\varepsilon,\varepsilon)$.

Next, we turn to DTC formulas. Consider the formula $\mathrm{DTC}[\varphi(x,y)](\varepsilon,\varepsilon)$. We construct a deterministic TWA over k-ary trees accepting exactly the k-ary trees the above formula defines. As in the case before, the automaton first evaluates all sentences χ in the Gaifman normal form of φ. Let m be the maximum number of vertices occurring in an r-neighborhood of a tree. That is, $m := \max\{|S_r^t(u)| \mid t \text{ a tree}, u \in \mathrm{dom}(t)\}$. For each isomorphism type τ of r-neighborhoods with one distinguished vertex, the automaton additionally computes the number of occurrences of τ in t up to $m+2$. That is, M computes the function $f_{r,m+2}^t$ as specified in the proof of Proposition 1. Now, given a u, to find a v such that $\varphi(u,v)$ holds, the automaton proceeds as follows. Let Y_0 be the set of all vertices w in $N_{2r}^t(u)$ such that $N_r^t(u,w) \models \varphi(x,y)$. By inspecting the $3r$-neighborhood of u, M can compute $|Y_0|$.

Next, it computes the type of $N_r^t(u)$ and the set T of all types τ of r-neighborhoods for which the following holds: if $N_r^t(w)$ is of type τ and $N_r^t(w)$ and $N_r^t(u)$ are disjoint then $N_r^t(u,w) \models \varphi(x,y)$. The latter is a fixed finite computation which can be encoded into the transition function. We call vertices of a type from T *good* vertices (w.r.t. the current u) and denote the set of good vertices in t by Y_1. Note that M can deduce $|Y_1|$ (up to $m+2$) from the precomputed information.

The set of good vertices in $N_{2r}^t(u)$ that are not in Y_0 is denoted by Y_2. By inspecting the $3r$-neighborhood of u again, M computes $|Y_2|$ and the relative positions of all vertices in Y_2 w.r.t. u.

Let $Y = Y_0 \cup (Y_1 - Y_2)$. Y is the set of vertices w of t such that $N_r^t(u,w) \models \varphi(x,y)$. M can compute $|Y|$ without further moving (note that $|Y_2| \leq m$). If $|Y|$ is different from 1 then M can immediately reject. Let us assume in the following that $|Y| = 1$.

If the unique element v of Y is from Y_0, M can directly go to v. If, on the other hand, $Y_0 = Y_2 = \emptyset$ then M can move to the unique $v \in Y_1$ via a DFS traversal of the tree.

The only complicated case is when $Y_0 = \emptyset$ and $|Y_1 - Y_2| = 1$ but $Y_2 \neq \emptyset$. The complication arises from the following possibility. M has to traverse the tree to find the correct unique good w outside $N^t_{2r}(u)$. As it does not know[2] in which part of the tree w is located its way to w might lead back to u. In that case we have to make sure that it does not confuse w with a vertex from Y_2.

We can assume w.l.o.g. that the root of the tree is not in $N^t_{3r}(u)$ because otherwise M can easily distinguish the vertices of Y_2 from the desired vertex. Now, M proceeds as follows. It starts a DFS walk through the tree starting from u and first inspecting the first subtree of u. Whenever it encounters a vertex z it starts a subcomputation which inspects the $3r$-neighborhood of z to find out whether there is a good vertex w in $N^t_{2r}(z)$. If such a w is found then M checks whether there is a vertex u' such that w has the same relative position w.r.t. u' as one of the vertices in Y_2 had wrt u and whether there are good vertices w' in the same relative position to u' as there are vertices in Y_2 relative to u. If M finds a w for which either no such u' exists, or u' is behind z in the DFS order then w is the desired vertex and M goes there. Otherwise it proceeds in its DFS walk. When the DFS walk finishes at the root (without finding the target vertex) then M walks back (reverse DFS) until it reaches u again (easily recognized as the first vertex w which has vertices from Y_1 in its neighborhood and relates to them as u does). Then it starts a reverse DFS walk from u (going upwards first) analogously to the first DFS walk.

We have to show that M always finds the correct target vertex. First, we show that M never moves to a vertex in Y_2. Let $w \in Y_2$. If M reaches w during the inspection of the neighborhood of a vertex z before its DFS walk arrives at the root then it will find out that it relates to u as a Y_2-vertex and that u is in the DFS order before z. The analogous statement holds true if z is found in the reverse DFS walk after coming back to u.

Finally, we show that M indeed reaches a target vertex v from Y_1 (which then is the correct one). This follows easily from the fact that either v does not relate to any u' as the Y_2 vertices do to u or this u' is different from u. Hence, it is encountered either in the DFS walk or in the reverse DFS walk. □

4 Weakness of tree-walking automata

Let k be fixed and let T_k be the set of all k-ary trees the leaves of which are labeled with 0 or 1. For each vertex v of a tree $v \in T_k$ we inductively define a value 0 or 1 as follows. If v is a leaf then its value (0 or 1) is determined by its label (0 or 1). If v has i children then v has the value 1 if and only if at least $\frac{i}{2}$ of its children have the value 1. Intuitively, T_k is the set of all tree-structured Majority circuits. Let T^1_k (T^0_k) denote the set of all trees $t \in T_k$ for which the root gets the value 1 (0).

We call a vertex v of a tree $t \in T_k$ a *1-vertex* (*0-vertex*) if it gets the value 1 (0). Analogously, we call the subtree rooted at a 1-vertex (0-vertex) a *1-subtree* (*0-subtree*).

Lemma 2. *For each k, the set T^1_k is a regular set of trees.*

Proof. Whether a tree from T_k is in T^1_k can be easily expressed by an MSO formula $\exists X \varphi$, where X is intended to be the set of vertices that have value 1 and φ is a first-order formula which checks that the root of a tree is in X and whether X is consistent with the bottom-up evaluation described above. □

We have seen in Example 1 that there is a deterministic TWA which recognizes T^1_2 (note that $T^1_2 \cup T^0_2$ can be seen as the set of trees representing Boolean circuits consisting of only OR-gates). This was due to the fact that the automaton, after evaluating a right subtree of a vertex v, could conclude the value of the corresponding left subtree of v from the label of v and the fact that it had to enter the right subtree. For $k > 2$, things are more complicated. In fact, we conjecture the following.

[2] Actually, there is an alternative construction where M keeps track of the relative positions of neighbourhoods like $N^t_r(u)$ and $N^t_{2r}(v)$ as these may only appear a bounded number of times in t (otherwise Y would be too large).

Conjecture 1. For $k > 2$, T^1_k can not be recognized by a deterministic or non-deterministic TWA.

In this section we prove this conjecture fora restricted type of TWAs, 1-bounded TWAs. The proof can be easily generalized to the sets T^1_k, for each $k > 3$. Furthermore, it can be extended to show that, for each r there is a related regular set of trees which can not be recognized by any r-restricted TWA.

Before we state and prove the result we introduce some important concepts for that proof and show a purely combinatorial result.

For any $d \geq 1$ a *critical tree* of depth d is a full ternary tree t of depth d with the following properties.

- $t \in T^1_3$;
- each 1-vertex of t has exactly two children which are 1-vertices;
- each 0-vertex has only 0-vertices as children.

In particular, there are no 1-leaves in 0-subtrees of t. Intuitively, a critical tree of depth d is a tree of depth d from T_3 which contains a full binary subtree of depth d which only has 1-leaves and all other leaves are 0-leaves. In particular, a critical tree of depth d has 2^d 1-leaves.

A *numbering* N of a critical tree t of depth d is an injective mapping of the 1-leaves of t into the set $\{0, \ldots, 2^d - 1\}$. All critical trees of a fixed size d are defined on the same tree domain τ_d. A mapping M which maps each leaf of τ_d to a subset of $\{0, \ldots, 2^d - 1\}$ with at most m elements is called an *m-labeling.* We say that an m-labeling M of τ_d is *compatible* with a numbering N of a critical tree t of depth d if, for each 1-leaf v of t, $N(v) \in M(v)$.

Lemma 3. *For each $m > 0$ there is a $d > 0$ such that, for each m-labeling M of τ_d there is a critical tree of depth d for which there is no numbering that is compatible with M.*

Proof. Let d be chosen such that $\left(\frac{4}{3}\right)^d > m$. Let M be an m-labeling of τ_d. As, for each leaf v of τ_d, $M(v)$ contains at most m elements from $\{0, \ldots, 2^d - 1\}$ there must be some $i \in \{0, \ldots, 2^d - 1\}$ which occurs in $M(v)$ for at most $\frac{m3^d}{2^d}$ leaves v of τ_d. Let A be the set of vertices v such that $i \in M(v)$. We construct a critical tree t of depth d which does not have a 1-leaf from A. This implies the statement of the lemma as t can not have a compatible numbering because no leaf of t can be numbered by i.

Let w_0, w_1, w_2 be the children of the root and let, for each $i \in \{0, 1, 2\}$, $A_i := \{v \in A \mid v \text{ is a leaf in } t_{w_i}\}$. Let w_0 be chosen such that $|A_0|$ is maximal. Clearly, both A_1 and A_2 contain at most $\frac{|A|}{2}$ elements from A. In t, w_0 will be a 0-vertex and w_1 and w_2 will be 1-vertices. Hence, each subtree rooted at 1-vertices of depth 1 contains at most $\frac{|A|}{2}$ elements from A. We proceed inductively in an analogous manner for these subtrees. At each depth i we will select the 2 children of 1-vertices that have the least number of leaves from A in their subtree. Hence, for each j, the selected 1-vertices of t at depth j may contain at most $\frac{|A|}{2^j}$ 1-leaves from A in their subtree. In the end, we have constructed a critical tree which has at most

$$\frac{|A|}{2^d} = \frac{m3^d}{2^d 2^d} < 1$$

1-vertices from A in the subtree rooted at a leaf. Hence, t has no 1-leaves from A at all. $\square$

Theorem 2. *(a) There is no 1-bounded (deterministic or non-deterministic) TWA which recognizes T^1_3.*
(b) For each $r > 0$, there is is a regular tree language that can not be recognized by an r-restricted (deterministic or non-deterministic) TWA.

Proof. The main task is to prove (a). Statement (b) will follow by an easy generalization of that proof.

Towards a contradiction assume there exists a non-deterministic TWA M' which recognizes T^1_3. The proof consists of 3 main steps:

– We transform M' into a TWA M which accepts exactly the same trees as M' but always rejects when it visits a 0-leaf. From this property we can conclude that if M accepts a certain tree t then it also accepts every tree which results from t by replacing 0-subtrees with 1-subtrees.
– We show that there are two trees $t, t' \in T_3^1$ with the same tree domain such that M has accepting computations for these trees which enter some vertex v in the same state s but with different "histories".
– We show these these accepting computations can be combined into one accepting computation on a tree from T_3^0.

Before we describe the construction of M we show that the 0-subtrees and 1-subtrees have disjoint sets of behaviour functions. Let, for $i \in \{1, \dots, 3\}$, $F^1_{M',i}$ ($F^0_{M',i}$) denote the set $\{f_{M',t,i} \mid t \in T_3^1\}$ ($\{f_{M',t,i} \mid t \in T_3^0\}$) of behaviour functions that M' can have on 1-subtrees (0-subtrees) that have child number i. If, for some i, $F^1_{M',i} \cap F^0_{M',i} \neq \emptyset$ then we can easily construct trees $t_1 \in T_3^1$ and $t_0 \in T_3^0$ that are either both accepted or both rejected by M'. To this end let $f \in F^1_{M',i} \cap F^0_{M',i}$ and let $t^0 \in T_3^0$ and $t^1 \in T_3^1$ such that $f_{M',t^0,i} = f_{M',t^1,i}$. Let t_0 be a tree which consists of a root with 3 children which has t^0 as i-th child and one of the other children is a 1-leaf and the other is a 0-leaf. The tree t_1 is the same as t_0 with the only exception that it has the subtree t^1 instead of t^0. As the behaviour functions of M' on t^0 and t^1 are the same the acceptance behaviour of M' on t_0 and t_1 is also the same.

Hence, we can assume that $F^1_{M',i}$ and $F^0_{M',i}$ are disjoint.

We turn to the construction of M. Intuitively, the idea is as follows. Whenever M' can go from a vertex v to vi then M can either do the same or, to prevent visiting a 0-leaf, it can guess that vi is the root of a 0-subtree. In the latter case instead of going down to vi it picks a behaviour function $f \in F^0_{M',i}$ and enters a new state at v according to f. Formally, for each $j \le k$, $s \in S$ and $\sigma \in \Sigma - \{0\}$,

$$\delta^i_M(j,s,\sigma) = \delta^i_{M'}(j,s,\sigma) \cup \bigcup_{\substack{(j',s') \in \delta^i_{M'}(j,s,\sigma) \\ f \in F^0_{M',j'}}} (0, f(s')).$$

For each $j \le k$ and $s \in S$, we define $\delta^i_M(j,s,0) = (0,\perp)$ where $\perp$ is a state from which no transition is possible.

We have to show that M accepts exactly all trees in T_3^1. Let therefore t be from T_3^1, hence, by assumption, t is accepted by M'.

Let $C' = c'_0, c'_1, \dots, c'_n$ be an accepting computation of M' on t. We show that there is also an accepting computation $C = c_0, \dots, c_m$ of M on t. We construct C by suitably modifying C'. Let $v = wj$ be a 0-leaf of t, for some vertex w and some j. By the prerequisite on M', v is at most visited once in C'. If v is not visited at all, we do not need to modify C' with respect to v. If v is visited then there are successive configurations $[w, s_i], [v, s_{i+1}], [w, s_{i+2}]$ in C'. Here, we assume w.l.o.g. that M' moves in each step. In C we replace these 3 configurations by $[w, s_i], [w, s_{i+2}]$. This reflects a legal transition of M as it corresponds to the computation of M' on the 0-subtree which consists of a single 0-leaf. We end up with a legal accepting computation C on t.

For the opposite direction, let $C = c_0, \dots, c_m$ be an accepting computation of M on a tree $t \in T_3$. We construct a tree t' by replacing some of the 0-leaves of t by 0-trees and an accepting computation C' of M' on t' by modifying C accordingly. By the construction of t' it will follow that $t \in T_3^1$ if and only if $t' \in T_3^1$. Hence, as M' accepts t' it also accepts t.

More formally, let $[v, s], [v, s']$ be two successive configurations from C such that it does not hold $[v, s] \Rightarrow_{M',t} [v, s']$. Hence, this subcomputation is possible only by the new transitions introduced in M. By definition, there must be $j \le k$, $s'' \in S$, $f \in F^0_{M',j}$ and a 0-tree t_0 such that $[v, s] \Rightarrow_{M',t} [vj, s'']$ and $f_{M',t_0,j}(s'') = s'$. From this we can conclude that there exist configurations $c(v,1), \dots, c(v,l)$ such that $[v,s], [vj,s''], c(v,l), \dots, c(v,l), [v,s']$ is a legal subcomputation of M' on the tree, denoted by $t(v)$, in which the subtree rooted at vj

is replaced by the 0-subtree t_0. If $t \in T_3^0$ then also $t(v) \in T_3^0$ as we only replaced a (0- or 1-) subtree by a 0-subtree. By inductively applying this argument we arrive at a tree t' and an accepting computation C' on t'. Hence, by assumption, $t' \in T_3^1$ and therefore $t \in T_3^1$.

It should be noted that the latter construction relies on the assumption that M' traverses each edge at most once. Otherwise, it could be the case that the configuration C visits v two times but the extension to C' makes use of two different 0-subtrees rooted at v.

Let now t be a critical tree of depth d and let $C = c_0, \ldots, c_n$ be an accepting computation of M on t. If C does not visit all 1-leaves then we can easily construct a tree $t' \in T_3^0$ which is accepted by M. Hence, we assume that C visits each 1-leaf of t exactly once. Let v be such a 1-leaf of t and let $c_j = [v, s]$ be the configuration of C which visits it. Let, for $i \in \{1, \ldots, d\}$, v_i denote the vertex of depth i on the path from the root of t to v. As v is a 1-leaf, each v_i is a 1-vertex. Therefore, as t is critical, each vertex v_i has exactly one sibling v_i' which is a 1-vertex. For each i, v_i' is visited in exactly one of the subcomputations $c_0, \ldots, c_{j-1}$ and $c_{j+1}, \ldots, c_m$. We define, for each 1-leaf v its *history string* $z = h_{t,C}(v) = z_1 \cdots z_d$ by setting $z_i = 1$ if and only if v_i' is visited in $c_0, \ldots, c_{j-1}$. These history strings have a couple of nice properties which are straightforward to prove given the assumptions on M.

- The binary number $b_{t,C}(v)$ represented by $z = h_{t,C}(v)$ (where z_d is the least significant bit) coincides with the number of 1-leaves that are visited in C before v. This follows from the restriction that the automaton can visit each subtree at most once. Hence, when a 1-vertex v at level i is entered, then all 2^{d-i} 1-leaves in the subtree of v have to be visited before the subtree is left. In this way, a 1 at the i-th bit of the history string corresponds to 2^{d-i} 1-leaves that have been already visited.
- Consequently, the function $b_{t,C}$ defines a numbering of t. In particular, each 0-1-string occurs exactly once as a history string of a 1-leaf v.

We claim that, there exists a d, two critical trees t, t' of depth d, two accepting computations C, C' of M on t, t', respectively, and a leaf v of τ_d such that

- v is a 1-leaf of t and t',
- C and C' visit v in the same state s, and
- $h_{C,t}(v) \neq h_{C',t'}(v)$.

Towards a contradiction assume that this claim is false. Let m be the number of states of M and let d be a number as given by Lemma 3. Let t and t' be two critical trees of depth d, let v be a common 1-leaf of t and t' and let C and C' be accepting computations of t and t', respectively, which visit v in the same state s. The assumption implies that $h_{C,t}(v) = h_{C',t'}(v)$ and therefore $b_{C,t}(v) = b_{C',t'}(v)$. We can conclude that, for each vertex v of τ_d and each state s of M, there is only one number $n(v, s)$ such that $b_{C,t}(v) = n(v, s)$ for all critical trees t with 1-leaf v and all accepting computations which visit v in the state s. In other terms, there exists an m-labeling M of the leaves of τ_d such that, for each critical t and each accepting computation C on t the numbering $b_{C,t}$ is compatible with M. This contradicts Lemma 3, as desired. Therefore, the claim is proved.

Let d, t, t', C, C' and v be as given by the above claim. We complete the proof by constructing a tree $t_0 \in T_3^0$ which is accepted by M. Let $z = h_{C,t}(v)$ and $z' = h_{C',t'}(v)$. Let j be minimal such that $z_j \neq z_j'$. We can assume w.l.o.g. that $z_j = 0$ and $z_j' = 1$. Let, for each $i \in \{1, \ldots, d\}$, v_i be defined as above, w_i be the 1-sibling of v_i in t and w_i' be the 1-sibling of v_i in t'. We construct t_0 as follows.

- For each $i < j$, if $z_i = 1$ (i.e., w_i is visited before v in C) then we copy the subtrees rooted at the siblings of v_i from t.
- For each $i < j$, if $z_i = 0$ (i.e., w_i' is visited before v in C') then we copy the subtrees rooted at the siblings of v_i from t'.
- At the siblings of v_j we root 0-subtrees. This assures that t_0 is a 0-tree.

– In the subtree rooted at v_j all leaves are labeled 1.

Let $C = c_0, \ldots, c_m$, $C' = c'_0, \ldots, c'_n$ and let k and k' be such that $c_k = c'_{k'}$ are the configurations in which v is visited.

It is straightforward to check that

- $c_0, \ldots, c_k$ is a valid subcomputation on t_0 because all 1-leaves of t that are visited in $c_0, \ldots, c_k$ are also 1-leaves in t_0;
- $c'_{k'}, \ldots, c'_n$ is a valid subcomputation on t_0 because all 1-leaves of t' that are visited in $c'_{k'}, \ldots, c'_n$ are also 1-leaves in t_0; hence
- $c_0, \ldots, c_k = c'_{k'}, \ldots, c'_n$ is an accepting computation on t_0, the desired contradiction.

This concludes the proof of statement (a).

To prove (b) we use a slightly different set U_3^1 of trees. These trees have an additional label, $+$. Inner vertices that are labeled with $+$ have 3 children which are interpreted as threshold gates. Inner vertices that are not labeled with $+$ have only 1 child. Hence, at a $+$-vertex there are starting 3 paths which lead either to another $+$-vertex or to a leaf. Now a $+$-vertex is evaluated to 1, if at least 2 of its 3 descendants ($+$-vertex or leaf) evaluate to 1. Intuitively, U_3^1 is the same as T_3^1 but the edges of trees in T_3^1 are replaced by paths in U_3^1. In fact, the proof of the fact that no r-restricted TWA recognizes U_3^1 is almost word for word the proof given in (a) but in the trees that are used, each edge has to be replaced by a path of length $r+1$. The old vertices are labeled with $+$, the new ones not. □

Acknowledgements.

We thank Joost Engelfriet, Clemens Lauteman, and Martin Grohe for helpful discussions.

References

[ABS99] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web : From Relations to Semistructured Data and XML.* Morgan Kaufmann, 1999.

[AU71] A. V. Aho and J. D. Ullman. Translations on a context-free grammar. *Inform. and Control*, 19:439–475, 1971.

[BMN99] G. Bex, S. Maneth, and F. Neven. XSL revisiTed:variables add power. Submitted.

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

[BHW99] A. Brüggeman-Klein, S. Hermann, and D. Wood. Context, caterpillars, tree automata, and tre pattern matching. In Proceedings of the 4th International Conference on Developments in Language Theory, 1999.

[Cla99] J. Clark. XSL Transformations version 1.0 (november 1999). http://www.w3.org/TR/xslt.

[EF95] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 1995.

[EH99a] J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In J. Karhum ki, 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.

[EH99b] J. Engelfriet and H. J. Hoogeboom. Private communication.

[EHvB99] J. Engelfriet, H.J. Hoogeboom, and J.-P. van Best. Trips on trees. *Acta Cybernetica*, 14:51–64, 1999.

[FM] Z. Fülöp and S. Maneth. Domains of partial attributed tree transducers. Technical Report 99-08, Leiden University, 1999.

[Ga82] H. Gaifman. On local and nonlocal properties. In J. Stern, editor, *Logic Colloquium '81*, pages 105–135. North Holland, 1982.

[GS97] F. Gécseg and M. Steinby. Tree languages. In Rozenberg and Salomaa [RS97], chapter 1.

[KS81] T. Kamimura and G. Slutzki. Parallel and two-way automata on directed ordered acyclic graphs. *Information and Control*, 49(1):10–51, April 1981.

[MSV] Y. Bargury and J. A. Makowsky. The expressive power of transitive closure logic and 2-way multi-head automata. In E. Borger, G. Jäger, H. K. Büning, and M. M. Richter editors. *Computer Science Logic*, volume 626 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 1991.

[MSV] T. Milo, D. Suciu, and V. Vianu. Type checking for XML transformers. Submitted.

[N99] F. Neven. *Design and Analysis of Query Languages for Structured Documents — A Formal and Logical Approach.* Doctor's thesis, Limburgs Universitair Centrum (LUC), 1999.

[NS] F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. Submitted.

[Pot94] A. Potthoff. Logische Klassifizierung regulärer Baumsprachen. Doctor's thesis, Institut für Informatik u. Prakt. Math., Universität Kiel, 1994.

[RS97] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 3. Springer, 1997.

[Tho97a] W. Thomas. Languages, automata, and logic. In Rozenberg and Salomaa [RS97], chapter 7.