One-Visit Caterpillar Tree Automata

A. Okhotin, K. Salomaa, M. Domaratzki
{okhotin,ksalomaa,domaratz}@cs.queensu.ca

Technical report 2002–459
School of Computing, Queen's University,
Kingston, Ontario, Canada K7L 3N6

August 2002

Abstract

We study caterpillar tree automata [3] that are restricted to enter any subtree at most one time (or k times). We show that, somewhat surprisingly, the deterministic one-visit automata can already, for instance, evaluate trees where the nodes represent some non-associative operations. We show that there exist regular tree languages that cannot be accepted by a one-visit automaton, thus proving a weakened form of a conjecture of Brüggemann-Klein and Wood [3]. We establish that the k-visit property is decidable.

1 Introduction

Recently formal properties and grammatical characterizations of SGML and XML documents have been investigated in [1, 2, 3]. The work involves several classical language-theoretic methods. For fundamental issues concerning markup languages see also [10].

Caterpillar expressions and automata were introduced by Brüggemann-Klein and Wood [3] for applications in specifying style sheets for XML documents and in tree pattern matching. A caterpillar automaton moves around a tree using a sequence of atomic moves and tests defined by a context-mapping

of a regular language. In an atomic move the automaton can move from the current node of the tree to its parent or to one of the children. It is shown in [3] that tree languages accepted by caterpillar automata are regular. The authors conjecture that this inclusion is strict.

Although it seems very likely that caterpillar automata cannot accept all regular tree languages, we show that they can perform some surprising computations. We show, for example, that caterpillars can evaluate trees where the internal binary nodes represent addition and subtraction modulo k. In all the example "caterpillar algorithms" considered here and in [3] it is sufficient for the automaton to visit any given subtree at most once. In fact, usually the computations can be performed by traversing the tree depth-first in left-to-right order. These observations motivate our definitions of left-to-right (LRCAT) and one-visit (1-CAT) caterpillar automata.

It is fairly easy to see that LRCAT automata cannot accept all regular tree languages. We prove that there exist regular tree languages that cannot be accepted by any deterministic one-visit automaton. The same construction establishes that two-visit caterpillars are strictly more powerful than the 1-CAT automata. We do not know whether the (deterministic or nondeterministic) k-visit hierarchy is strict with respect to k. Note that any accepting computation of a fixed caterpillar automaton can visit any subtree at most a constant number of times (where the constant depends on the automaton). We show that given a caterpillar automaton A it is decidable whether A has the one-visit property or whether A admits a loop.

The conjecture from [3] that the family of tree languages accepted by general caterpillar automata is strictly included in the regular tree languages remains open. It should be noted that [7, 8, 9] consider finite tree-walking automata and show that the nondeterministic tree-walking automata are strictly more powerful than the deterministic ones and the tree languages accepted by the nondeterministic variant are strictly included in the regular tree languages. However, an essential difference to caterpillar automata is that when making an upward move the tree-walking automata of [7, 8, 9] cannot determine from which descendant of the target node the move originated. Because of this the tree-walking automata can be shown to be unable to traverse all nodes of a balanced tree of sufficient height. On the other hand, caterpillar automata can systematically traverse all nodes of an arbitrary tree [3]. The alternating tree-walking automata accept exactly the regular tree languages [9]. On the other hand, parallel two-way tree automata also do not recognize more than the regular tree languages [4].

The organization of the paper is as follows. In the next section we recall some basic definitions concerning trees, define the caterpillar tree automata, as well as present several examples. For ease of presentation we slightly modify the original definition of caterpillars from [3]. In the third section we discuss the one-visit caterpillar automata and the last section describes some open problems.

2 Caterpillar tree automata

We assume the reader to be familiar with finite tree automata and only briefly recall the definitions needed here. For all unexplained notions the reader may consult [5, 6].

The cardinality of a finite set S is $\#S$. For $m \in \mathbb{N}$ we denote $[m] = \{1, \ldots, m\}$.

A ranked alphabet is a pair (Σ, r) where Σ is a finite set and $r : \Sigma \longrightarrow \mathbb{N} \cup \{0\}$ is a function that associates with each element $\sigma \in \Sigma$ its rank $r(\sigma)$. The set of elements of rank m is Σ_m, $m \geq 0$. Instead of (Σ, r), we will usually speak of the ranked alphabet Σ and assume that r is known. The set of trees (or terms) over Σ, F_Σ is the smallest set S satisfying the condition: if $m \geq 0$, $\sigma \in \Sigma_m$ and $t_1, \ldots, t_m \in S$ then $\sigma(t_1, \ldots, t_m) \in S$.

We assume that notions such as the *root,* a *leaf,* a *subtree* and the *height* of a tree are known. We use the convention that the height of a single node tree is zero. Let t be a tree and u some node of t. The subtree of t at node u is denoted t_u. The tree obtained from t by replacing the subtree at node u with a tree s is denoted $t[u \leftarrow s]$.

A nondeterministic bottom-up tree automaton is a construct $A = (\Sigma, Q, Q_F, g)$ where Σ is a ranked alphabet, Q is a finite set of states, $Q_F \subseteq Q$ is a set of accepting states and g associates to each $\sigma \in \Sigma_m$ a mapping $\sigma_g : Q^m \longrightarrow 2^Q$, $m \geq 0$. For each $t = \sigma(t_1, \ldots, t_m) \in F_\Sigma$ we define inductively the set $t_g \subseteq Q$ by setting $q \in t_g$ if and only if there exist $q_i \in (t_i)_g$, $i = 1, \ldots, m$, such that $q \in \sigma_g(q_1, \ldots, q_m)$. (Intuitively, t_g consists of the states of Q that A may reach at the root of t.) The tree language accepted by A is $L(A) = \{t \in f_\Sigma \mid t_g \cap Q_F \neq \emptyset\}$. The nondeterministic (bottom-up or top-down) tree automata accept the family of *regular tree languages* [5, 6].

Next we define the caterpillar tree automata. A caterpillar consists of a single (read-only) finite state control that can make atomic moves either to the parent (up-move) or to one of the children (down-move) of the current

node. The original definition of caterpillars in [3] was motivated by context specification and evaluation in markup languages and for this reason the definition there uses a regular language to control the movements of (i.e., define a context mapping for) a caterpillar. We are mainly interested in proving properties of tree languages accepted by caterpillar automata and for our purposes it is more convenient to directly specify a transition relation that gives the next move(s) of the automaton in its current state.

A difference to [3] is that we consider only trees where the nodes have a bounded number of children. This makes the notations simpler since we can use a ranked alphabet to label nodes of the trees. In terms of our main result (which is a negative result stating that there exist regular tree languages that cannot be accepted by certain type caterpillars) this restriction clearly does not cause any loss of generality. Also when the nodes of the trees have bounded arity, it is natural to allow the caterpillars to make down-moves to any of the descendants. Then we do not need moves to the siblings of the current nodes and also can eliminate test instructions that determine whether the current node is the leftmost or the rightmost descendant of its parent. In our definition an up-move implicitly determines this information. Note that in the original definition given in [3], a down-move can be made only to the leftmost or the rightmost child and then the automaton can move from one sibling to the next (or previous) sibling, as well as, test whether it is located in the leftmost or the rightmost sibling.

Another minor difference is that the original definition allows the caterpillar to test when it is at the root of the tree. In order to avoid additional notations we do not include this capability, and instead assume that the input tree is augmented with a special unary *stump*, as illustrated in Figure 1. Adding the stump to a tree is basically the same as adding begin and end markers to words, and has the advantage of being a more uniform tool than an explicit root predicate.

Definition 2.1 *Let Σ be a ranked alphabet. Then the augmented alphabet Σ' is defined as $\Sigma' = \Sigma \cup \{stump\}$, where stump ($stump \notin \Sigma$) is a symbol of rank one, and consequently,*

$$\Sigma'_m = \begin{cases} \Sigma_m \cup \{stump\}, & \text{if } m = 1 \\ \Sigma_m & \text{otherwise} \end{cases} \tag{2.1}$$

Now, given an arbitrary tree $t \in F_\Sigma$, denote the tree t augmented with a stump as $t' = stump(t)$ (see Figure 1).

Figure 1: (a) An arbitrary tree t; (b) t augmented with stump, denoted as t'.

Definition 2.2 *Let Σ be a ranked alphabet and $M = \max\{m \mid \Sigma'_m \neq \emptyset\}$.*

A caterpillar (tree) automaton *is a tuple*

$$A = (\Sigma, Q, q_0, Q_F, f) \tag{2.2}$$

where Q is the finite set of states, $q_0 \in Q$ is the initial state, $Q_F \subseteq Q$ is the set of accepting states, and

$$f : \Sigma' \times Q \to 2^{\{\text{down}\}\times\{1,\ldots,M\}\times Q \,\cup\, \{\text{up}\}\times Q^M} \tag{2.3}$$

is the state transition function, in which (i) elements of the form (down, i, q) instruct the automaton to go to the i-th descendant of the current node and switch to state q, while (ii) an element of the form $(\text{up}, q_1, \ldots, q_M)$ means that the automaton should go to the predecessor of the current node (which has the current node as the i-th descendant) and switch to the state q_i. Note that the new state after an up-move depends on the direction from where the automaton came from. For obvious reasons, it is required that $1 \leqslant i \leqslant r(\sigma)$ whenever $f(\sigma, q) \ni (\text{down}, i, q')$ and that $f(stump, q) \not\ni (\text{up}, q_1, \ldots q_m)$ for any $q, q_1, \ldots, q_m \in Q$.

The caterpillar A is said to be deterministic *if for any pair $(\sigma, q) \in \Sigma' \times Q$ it holds that $\#f(\sigma, q) = 1$; otherwise A is nondeterministic.*

For deterministic caterpillars it is convenient to redefine f as

$$f : \Sigma' \times Q \to \{\text{down}\}\times\{1,\ldots,M\}\times Q \;\cup\; \{\text{up}\}\times Q^M \tag{2.4}$$

Now we can define the computations of and the tree language accepted by a caterpillar automaton.

Definition 2.3 *Let Σ be a ranked alphabet. Let A be a caterpillar automaton. Define a ranked alphabet $\Omega = \Sigma' \cup (\Sigma' \times Q)$, and let where*

$$\Omega_m = \Sigma'_m \cup \{(\sigma, q) \mid \sigma \in \Sigma'_m, q \in Q\}.$$

An Ω-tree t is an A-configuration if t contains exactly one node labeled by an element of $\Sigma' \times Q$. The set of A-configurations is denoted $\mathrm{Config}(A)$. *The relation defining single-step computations of A, $\vdash_A \subseteq \mathrm{Config}(A) \times \mathrm{Config}(A)$, is defined as follows. Let $C, C' \in \mathrm{Config}(A)$. Then $C \vdash_A C'$ if and only if one of the following possibilities holds:*

(i) *C contains a subtree*

$$t = (\sigma, q)(t_1, \ldots, t_{i-1}, \tau(r_1, \ldots, r_n), t_{i+1}, \ldots, t_m),$$
$$\sigma \in \Sigma'_m,\ \tau \in \Sigma'_n,\ q \in Q,\ 1 \leq i \leq m$$

and C' is obtained from C by replacing t with

$$\sigma(t_1, \ldots, t_{i-1}, (\tau, q')(r_1, \ldots, r_n), t_{i+1}, \ldots, t_m),$$

where $(\mathrm{down}, i, q') \in f(\sigma, q)$. This represents a down-move of A to the ith child of the node where the reading head is in C.

(ii) *C contains a subtree*

$$t = \tau(t_1, \ldots, t_{i-1}, (\sigma, q)(r_1, \ldots, r_m), t_{i+1}, \ldots, t_n),$$
$$\sigma \in \Sigma'_m,\ \tau \in \Sigma'_n,\ q \in Q,\ 1 \leq i \leq n$$

and C' is obtained from C by replacing t with

$$(\tau, q_i)(t_1, \ldots, t_{i-1}, \sigma(r_1, \ldots, r_m), t_{i+1}, \ldots, t_n),$$

where $(\mathrm{up}, q_1, \ldots, q_M) \in f(\sigma, q)$. This represents an up-move of A to the parent of the node u where the reading head is in C. In this case we know that u is the ith child of its parent.

Let $t \in F_\Sigma$. The initial configuration *corresponding to t is*

$$\mathrm{init}(t) = (\mathit{stump}, q_0)(t).$$

Intuitively, this means that the caterpillar begins the computation at the stump of the augmented tree t'.

A configuration $C \in \mathrm{Config}(A)$ *is* accepting *if the root of* C *(corresponding to the stump of* t'*) is labeled by a symbol* $(stump, q)$ *where* $q \in Q_F$*. The tree language accepted by the caterpillar automaton* A *is*

$$L(A) = \{t \in F_\Sigma \mid \mathrm{init}(t) \vdash_A^* C \quad \textit{for some accepting configuration } C\}.$$

The class of deterministic (respectively, nondeterministic) caterpillar automata is denoted CAT (respectively, $NCAT$). The family of tree languages accepted by a (sub)class X of caterpillar automata is denoted $\mathcal{L}(X)$.

If A is a deterministic caterpillar, for any configuration C there exists at most one configuration C' such that $C \vdash_A C'$.

It is clear that the automata defined in Definitions 2.2 and 2.3 accept the same family of tree languages as the caterpillars of [3] when restricted to trees of bounded arity. In the original definition, the movement of the caterpillar is controlled by a sequence of instructions, and the set of such sequences is a regular language. Thus possible control sequences can be simulated by our model with a finite control together with a set of allowed transitions, and vice versa. We leave the details of the constructions needed for the simulations to the interested reader.

Furthermore, the automata given in Definition 2.2 could in a natural way be extended to operate on trees of unbounded arity if we allow moves from one sibling to the next or the previous sibling as in [3].

Proposition 2.4 *[3]* $\mathcal{L}(NCAT)$ *is included in the family of regular tree languages.*

It was conjectured in [3] that the above inclusion is strict. We believe that the conjecture is true, however it turns out that deterministic caterpillars can already perform fairly surprising computations. This indicates that it may be difficult to find a proof for the conjecture.

Let us consider several examples of quite nontrivial tree languages that can be recognized already by deterministic caterpillar automata in a single left-to-right traversal of a tree.

Example 1 *A caterpillar automaton for evaluation of Boolean expressions with conjunction, disjunction and negation.*

Let $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$, *where* $\Sigma_0 = \{0, 1\}$, $\Sigma_1 = \{\neg\}$ *and* $\Sigma_2 = \{\vee, \wedge\}$. *For every tree* $t \in F_\Sigma$, *define its value (0 or 1):*

$$v(0) = 0 \tag{2.5a}$$

$$v(1) = 1 \tag{2.5b}$$

$$v(\neg(t)) = \begin{cases} 1, & \text{if } v(t) = 0 \\ 0, & \text{if } v(t) = 1 \end{cases} \tag{2.5c}$$

$$v(\wedge(t', t'')) = \begin{cases} 1, & \text{if } v(t') = 1 \text{ and } v(t'') = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.5d}$$

$$v(\vee(t', t'')) = \begin{cases} 1, & \text{if } v(t') = 1 \text{ or } v(t'') = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.5e}$$

The task we shall now consider is to compute the value of a given tree, or, to formulate it as a decision problem, whether a given tree has value 1. Define the tree language $L = \{t \mid v(t) = 1\}$.

At the first glance, the problem can seem to be incomputable by deterministic caterpillar automata, because, once an automaton returns from the second subtree of a certain conjunction or disjunction node, it will have to recall the value of the first subtree, and finite memory is not enough to store the values of all such first subtrees. However, as we shall now demonstrate, this intuition is wrong, because the very position of the caterpillar in the tree can be used to store all the necessary information.

Let us construct a deterministic left-to-right caterpillar automaton to recognize the trees from L. The set of states is defined as $\{u\} \cup \{l, r\} \times \{0, 1\}$, where u means "just came from above", while every pair (l, b) or (r, b) ($b \in \{0, 1\}$) indicates that the automaton just came from left (right) descendant of the current node (in case of a unary node, let us call its descendant left) and computed the value b.

- Whenever the automaton enters a conjunction node from above in the state u, it goes directly into its first descendant. When it returns from the first subtree – i.e., whenever the automaton enters a conjunction node from below/left in the state $(l, 0)$ or $(l, 1)$ – it has computed the value of this subtree.

  If it is zero – i.e., the state is $(l, 0)$, – then the value of the conjunction is also clearly zero, and the automaton goes directly upward, returning 0 (see Figure 2(a)) and thus enters the state $(l, 0)$ or $(r, 0)$.

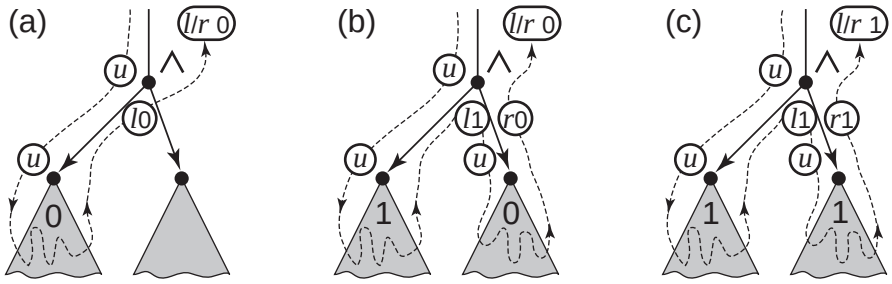

Figure 2: Evaluation of Boolean expressions: Treatment of conjunction.

If it is one, then the automaton goes to the right subtree. When it returns from there – i.e., whenever the automaton enters a conjunction node from below/right in the state $(r, 0)$ or $(r, 1)$, – it has computed the value of the right subtree. Since it could come into this subtree only in the case the left subtree has value one, the value of the right subtree is the value of the whole conjunction, and the automaton goes upward, returning this value (see Figure 2(b) and (c)) and entering the state $(l, 0)$ or $(r, 0)$ if it is now in the state $(r, 0)$, and the state $(l, 1)$ or $(r, 1)$ if it is now in $(r, 1)$.

- The same is done for a disjunction node: the left subtree is traversed first, and if it evaluates to 1, then the value 1 can be returned immediately (see Figure 3(a)); otherwise, the right subtree is traversed, and if it evaluates to 1, then 1 is returned (as in Figure 3(b)), and if it evaluates to 0, then 0 is returned (as in Figure 3(c)).

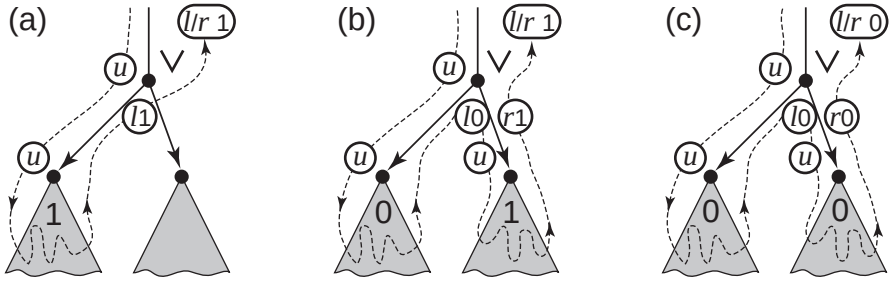

Figure 3: Evaluation of Boolean expressions: Treatment of disjunction.

- Whenever the automaton enters a negation node, it goes down, com-

putes the value of the subtree, and then negates that value and returns it upward: if the subtree evaluates to 0, then 1 is returned (see Figure 4(a)), and if the subtree evaluates to 1, then 0 is returned (see Figure 4(b)).

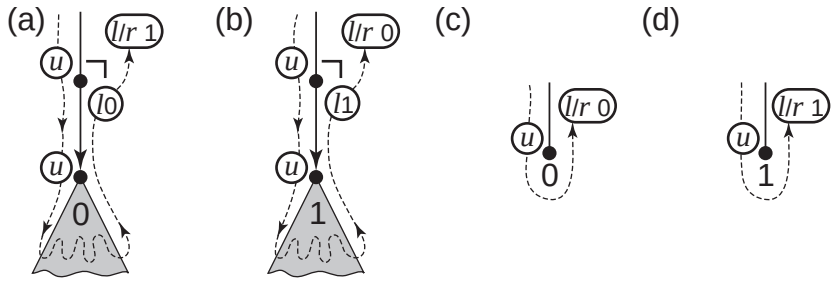

Figure 4: Evaluation of Boolean expressions: Treatment of negation and constants.

- Whenever the automaton enters a leaf labeled with 0 (1), it returns zero (one), as illustrated in Figure 4(c) and (d).

Now it suffices to define the initial state as u and set of accepting states as $\{(l, 1)\}$.

Example 2 *A caterpillar automaton for computing sum and difference modulo k.*

Fix an integer $k \geqslant 2$, and let $\Sigma = \Sigma_0 \cup \Sigma_2$, where the leaves $\Sigma_0 = \{0, \ldots, k-1\}$ are constant symbols, while the binary nodes $\Sigma_2 = \{\oplus, \ominus\}$ represent sum and difference modulo k respectively.

As in the previous example, we define the value of a tree from F_Σ as

$$v(i) = i \quad (\text{for all } i : \ 0 \leqslant i < k) \tag{2.6a}$$
$$v(\oplus(t', t'')) = v(t') + v(t'') \pmod{k} \tag{2.6b}$$
$$v(\ominus(t', t'')) = v(t') - v(t'') \pmod{k} \tag{2.6c}$$

Again, we formulate the task of computing the value of a tree as a decision problem of determining whether a given tree has some fixed value (let it be, for instance, 1), and look for a caterpillar automaton to recognize the tree language $L = \{t \mid v(t) = 1\}$.

Even in the simplest case of $k = 2$, where both $\oplus$ and $\ominus$ degrade to *exclusive or*, the problem cannot be solved using the methods of Example 1, because, once the automaton returns from the left descendant of a binary node, it should visit the right subtree regardless of the value of the left, while finite memory will definitely not be enough to store the values of all the traversed left subtrees. However, in this case a different very simple technique is applicable: since sum modulo 2 is associative, one can simply traverse all the leaves from left to right, storing the sum of the visited leaves in finite state control.

The case of sum and difference modulo k for $k \geqslant 3$ is not so trivial, since one of the operators is neither associative nor commutative. But still the above method can be extended for this case, if we notice that the value of a tree can be written as the sum of all its leaves in the left-to-right order, each with a plus or minus sign. For instance, the sample tree in Figure 5(a) denotes the expression

$$(4 \oplus 1) \ominus ((1 \oplus (2 \ominus 2)) \ominus (3 \ominus 1)) \tag{2.7}$$

which can be rewritten as

$$4 \oplus 1 \ominus 1 \ominus 2 \oplus 2 \oplus 3 \ominus 1 \tag{2.8}$$

A leaf is included with a positive sign if and only if the number of subtractions above is even – i.e., if the path from the root node to this leaf contains an even number of right arcs going out of $\ominus$ nodes. Thus it suffices to store this number modulo 2 in course of the computation and use it in every leaf to determine whether the leaf's value should be added to or subtracted from the accumulated sum.

Define the set of states as $Q = \{u, l, r\} \times \{0, \ldots, k-1\} \times \{+, -\}$, where the first component of every triple determines the direction, from which the automaton just came, the second component holds the current value of the sum (modulo k), and the third component records (modulo 2) the number of right $\ominus$ arcs on the path from the root to the current node, and consequently determines whether the values in the leaves are added to or subtracted from the sum.

The initial state of the automaton is $(u, 0, +)$. The accepting states are $(r, 1, +)$ and $(r, 1, -)$. The computation of this automaton for the case $k = 5$ on the sample tree is illustrated in Figure 5(b).

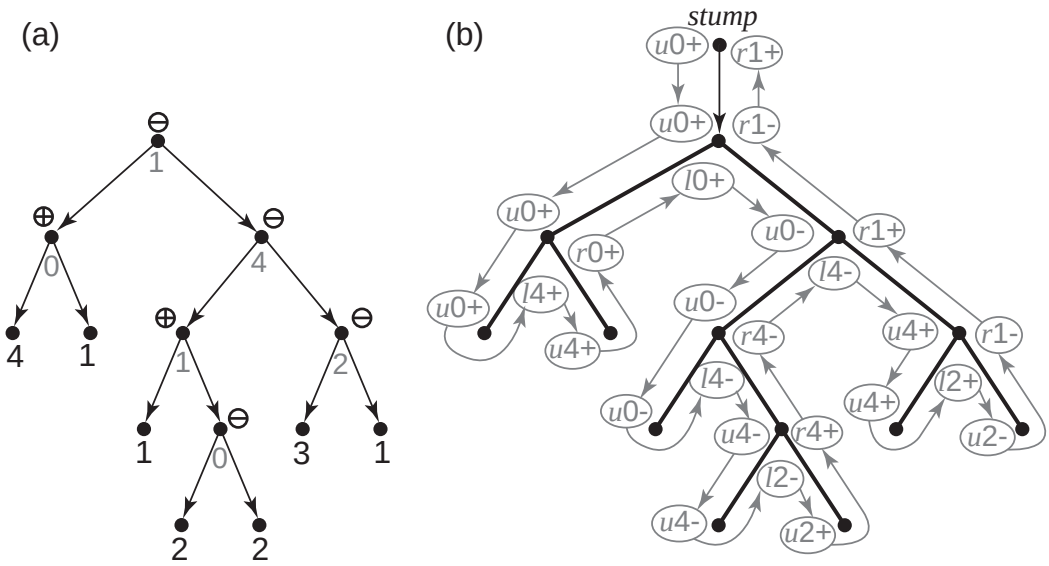

Figure 5: Computing sum and difference modulo $k = 5$: (a) A sample tree; (b) The computation on this tree.

3 One-visit automata

The computations of the caterpillars considered in the examples of the previous section are deterministic and, furthermore, traverse the input trees in a specified order that visits any subtree at most once. The same is true for the examples considered in [3]. At first sight it might seem possible that any caterpillar computation could be performed using a strategy that visits each subtree at most once. These observations motivate the following definitions of restricted types of caterpillar automata.

Definition 3.1 *Let A be a deterministic caterpillar automaton. Consider a computation of A on an input tree t and let s be a subtree of t. Each down-move to the root of s is said to be a* visit *to the subtree s.*

(i) *We say that A is a left-to-right automaton if A traverses each input tree in depth-first left-to-right order. The automaton is allowed to omit visiting some subtrees. The family of left-to-right automata is denoted LRCAT.*

(ii) *We say that A is a k-visit automaton, $k \geq 1$, if on any input tree t, A*

*visits any subtree of t at most k times. The family of k-visit automata is denoted k-*CAT*.*

Remark 1 *Our definition allows caterpillars to make a down-move to any of the children, whereas the original definition of [3] allows down-moves only to the leftmost and the rightmost child and then moves from one sibling to another. Thus a one-visit (or a k-visit) automaton as defined here could not be directly simulated by a caterpillar as originally defined that is allowed to visit any subtree only once (or k times). Of course when the nodes have bounded arity, also the original definition of caterpillars can in a natural way be modified to include down-moves to any of the children.*

A left-to-right caterpillar is a special case of a one-visit caterpillar. It is immediate that the transition relation of the caterpillar can be made to directly enforce a left-to-right strategy: the states of the automaton can simply contain an additional component that is used to guarantee that any input tree can be traversed only in left-to-right order. On the other hand, given an arbitrary caterpillar automaton A it is not necessarily immediately clear whether A satisfies the one-visit property (or more generally, the k-visit property). The following result guarantees that the k-visit property can be effectively determined.

Theorem 3.2 *Given $k \geq 1$ and an arbitrary deterministic caterpillar automaton $A = (\Sigma, Q, q_0, Q_F, f)$ we can decide whether or not A is a k-visit automaton.*

Proof sketch. We only give an outline of the proof which uses standard tree automata arguments. Assume that A is not a k-visit automaton and let t be a tree such that the computation of A on t violates the k-visit property. Let

$$\mathrm{init}(t) = C_0 \vdash_A C_1 \vdash_A \ldots \vdash_A C_m,$$

be the prefix (denoted by α_t) of the computation of A on t such that in the step $C_{m-1} \vdash_A C_m$ the caterpillar for the first time violates the k-visit property by making a down-move to a node n_0 for the $(k+1)$-st time. For each node u of t, let $\alpha_t[u]$ be the sequence

$$(q_1, p_1, q_2, p_2, \ldots)$$

where q_i is the state of A just after A made the ith down-move into u in α_t, and p_i is the state of A just before A made the ith up-move from the node u in α_t. (Note that the sequence may end either with a q_j or a p_j state.) For any node $u \neq n_0$, the length of the sequence $\alpha_t[u]$ is at most $2k$.

Consider any nodes u and u' of t such that the following holds

(i) $\alpha_t[u] = \alpha_t[u']$,

(ii) u' is a (not necessarily direct) descendant of u,

(iii) the node n_0 is not contained in t_u or it is contained in $t_{u'}$.

Then the automaton A has a computation also on $t[u \leftarrow t_{u'}]$ that violates the k-visit property. (Conditions (i) and (ii) guarantee that the substitution $u \leftarrow t_{u'}$ does not change anything in the part "outside of subtree t_u" or "inside of subtree $t_{u'}$". On the other hand, (ii), (iii) guarantee that the node n_0 does not "become deleted" in the substitution $u \leftarrow t_{u'}$.)

Thus as long as t has height greater than some constant depending only on the automaton A, we can always find distinct nodes u and u' of t that satisfy the above three conditions. In this case we can construct a strictly smaller tree $t' = t[u \leftarrow t_{u'}]$ such that the automaton A violates the k-visit property also on t'.

The above means that in order to check whether A satisfies the k-visit property it is sufficient to consider the computations of A on a finite set of trees S, where S can be effectively determined. ■

The above proof gives only a very inefficient brute-force method for testing the k-visit property. It would be interesting to know whether at least the one-visit property could be tested in polynomial time.

Note that even a deterministic caterpillar automaton may enter an infinite loop and this happens exactly then when it visits some subtree m times where m is greater than the number of states. As a consequence of Theorem 3.2 we have:

Corollary 3.3 *Given a deterministic caterpillar automaton A we can effectively decide whether or not A admits an infinite loop.*

The definition of the k-visit property could be extended in the natural way for nondeterministic caterpillars by requiring that any accepted tree is accepted by a computation that visits any subtree at most k times. However,

we do not know whether the decidability result of Theorem 3.2 would hold for nondeterministic automata.

We want to show that there exist regular tree languages that cannot be accepted by a deterministic one-visit automaton. First we construct a regular tree language that cannot be accepted by a left-to-right automaton. The below lemma is a direct consequence of Theorem 3.5 below and we present it only because the later result uses an extension of this simple proof.

Lemma 3.4 *There exist regular tree languages that do not belong to* $\mathcal{L}(LRCAT)$.

Proof. Choose $\Sigma = \Sigma_0 \cup \Sigma_3$ where $\Sigma_0 = \{0,1\}$ and $\Sigma_3 = \{f\}$. For any $w = i_1 i_2 \cdots i_m$, $i_j \in \{0,1\}$, $j = 1, \ldots m$, define the Σ-tree

$$t_w = f(i_1, f(i_2, \ldots, f(i_{m-1}, f(i_m, i_m, i_m), i_{m-1}) \ldots, i_2), i_1)$$

(see Figure 6(a)), and let

$$L = \{t_w \mid w \in \{0,1\}^+\}.$$

Clearly L is regular. For the sake of contradiction assume that L is accepted by a left-to-right caterpillar $A = (\Sigma, Q, q_0, Q_F, f)$.

Consider an arbitrary $t_w \in L$. A left leaf (respectively, a right leaf) of t_w is a leaf that is a left child (respectively, a right child) of some node labeled by f. The tree t_w has a unique middle leaf that is middle child of a node labeled by f. On any $t_w \in L$, A reaches the middle leaf after visiting all the left leaves and before visiting any of the right leaves.

Choose m so that $2^m > \#Q$. Thus there exist $w, w' \in \{0,1\}^m$, $w \neq w'$, such that A reaches the middle leaf of both t_w and $t_{w'}$ in the same state q. Let t be the tree obtained from t_w by replacing all the right leaves by the corresponding right leaves of $t_{w'}$. Since A enters the middle leaf of t in state q, A has to accept t. On the other hand, since $w \neq w'$ we have $t \notin L$. ■

It is clear that a one-visit caterpillar can accept the tree language L from the above proof by deterministically comparing the leftmost and the rightmost leaf at each level and then continuing to check the middle subtree. However, we can fool also an arbitrary one-visit automaton if the pairs of leaves have to be compared with respect to any possible path. This idea is formalized as follows.

Let $\Sigma = \Sigma_0 \cup \Sigma_3$, $\Sigma_3 = \{f\}$, be a ranked alphabet that contains one ternary symbol and some nullary symbols. We define the tree language $L(\Sigma)$ to consist of all Σ-trees that can be constructed using the below rules a finite number of times.

(i) For each $x \in \Sigma_0$, the tree $f(x,x,x)$ is in $L(\Sigma)$.

(ii) If $t \in L(\Sigma)$ and $x \in \Sigma_0$, then the trees

$$f(t,x,x), \quad f(x,t,x), \quad f(x,x,t)$$

are also in $L(\Sigma)$.

Each tree $t \in L(\Sigma)$ contains a unique path from the root to a node of height one such that all nodes of this path are labeled by the symbol f. This is called the *main path* of t, $mp(t)$. The two (or three, in the case of the last node) leaves connected to each node of $mp(t)$ have to be identical, see Figure 6(b).

The tree language $L(\Sigma)$ can be viewed as a generalization of the tree language constructed in the proof of Lemma 3.4 where the main path of a tree t can continue at any of the three childs of a given node. The idea is to prevent, on suitably chosen inputs, a deterministic automaton from "finding" the main path without looking at some of the leaves, and thus prevent a deterministic one-visit automaton from using a strategy analogous to the one mentioned after Lemma 3.4.

Theorem 3.5 *There exist regular tree languages that are not in* $\mathcal{L}$(1-CAT).

Proof. Choose $\Sigma = \Sigma_0 \cup \Sigma_3$, $\Sigma_0 = \{1,2,3,4,5,6,7\}$, $\Sigma_3 = \{f\}$, and let $L(\Sigma)$ be as above. Clearly $L(\Sigma)$ is regular so in order to prove the claim it is sufficient to show that $L(\Sigma) \notin \mathcal{L}$(1-CAT).

For the sake of contradiction assume that $L(\Sigma)$ is accepted by a deterministic one-visit automaton $A = (\Sigma, Q, q_0, Q_F, f)$. Without loss of generality we can assume that

$$\text{if } A \text{ accepts a tree } t, \text{ then } A \text{ visits all leaves of } t. \tag{3.9}$$

If the above were not true, A would clearly accept also trees not in $L(\Sigma)$. (The automaton A may reject inputs without visiting all the leaves.)

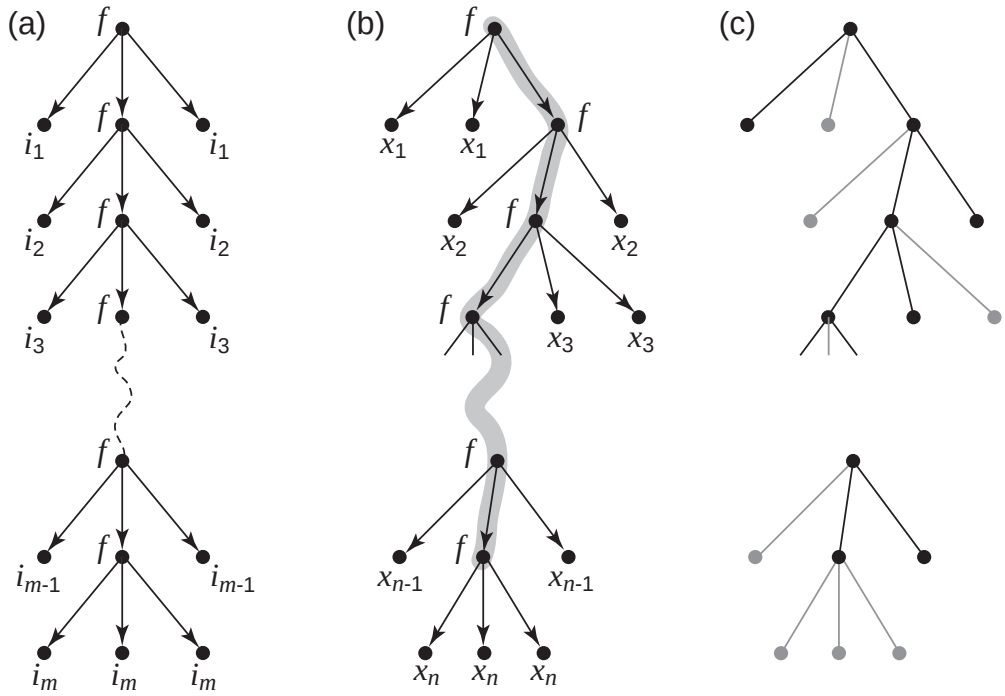

Figure 6: (a) A tree $t_{i_1 i_2 \ldots i_m}$ from Lemma 3.4; (b) A tree from Theorem 3.5 and its main path; (c) A skeletal tree corresponding to a possible descent.

To each tree $t \in L(\Sigma)$ we associate the *leaf-word* of t, $l_w(t) \in \Sigma_0^m$, where m is the height of t. If $t = f(x, x, x)$, $x \in \Sigma_0$, we define $l_w(t) = x$. If t is of the form $f(t', x, x)$, $f(x, t', x)$ or $f(x, x, t')$, $t' \in L(\Sigma)$, $x \in \Sigma_0$, we define

$$l_w(t) = x \cdot l_w(t').$$

Note that $l_w(t)$ is defined only for trees $t \in L(\Sigma)$.

Let $z \in \Sigma_0^m$, $m \geq 1$, be an arbitrary fixed word. We claim that there exists a unique tree $t_{(z)} \in L(\Sigma)$ such that both of the following conditions hold:

(i) $t_{(z)}$ has height m and $l_w(t_{(z)}) = z$.

(ii) The computation of A on $t_{(z)}$ reaches the last node of $mp(t_{(z)})$ labeled with f after visiting exactly one of the leaves associated with each of the earlier nodes of $mp(t_{(z)})$.

We construct $t_{(z)}$ as follows. The root of $t_{(z)}$ is labeled by f. Let $\{i_1, i_2, i_3\} = \{1, 2, 3\}$ and assume that when A is started in state q_0 at a node labeled by f it first makes a down-move to the i_1-th child. Then we choose that the i_1-th child is labeled by the first symbol of z. After this A necessarily makes an up-move followed by a down-move either to the i_2-th or i_3-th child, w.l.o.g. assume that this is the i_2-th child. We construct $t_{(z)}$ so that the main path $mp(t)$ continues at the i_2-th child of the root, and thus this node is labeled again by f. Note that A arrives at the i_2-th child in a state q that depends only on (the first symbol of) z.

Now the state q determines uniquely the child of the current node to which A makes a down-move. Property (3.9) implies that if $m > 1$, A has to make a down-move. We label this node with the second symbol of z. After this A has to make an up move, and then a down-move to a child that depends now only on the first two symbols of z.

Thus we see that $t_{(z)}$ can always be constructed so that (ii) holds and, since A is deterministic, for given z the construction is unique.

This kind of descent of A through a tree $t_{(z)}$ is depicted in Figure 6(c), in which the black nodes and arcs are those visited in course of the descent and the unvisited parts of the tree are shown in gray. The choice of nodes in such a descent can be encoded as a string

$$w_{(z)} \in \{(i, j) \mid i, j \in \{1, 2, 3\} \text{ AND } i \neq j\}^m, \tag{3.10}$$

and consequently there are no more than 6^m ways to go down through the trees from the set $\{t_{(z)} \mid z \in \Sigma_0^m\}$ constructed as above.

Choose k so that

$$7^k > 6^k \cdot (\#Q).$$

There exist 7^k words $z \in \Sigma_0^k$. There exist at most 6^k different ways to descend through the trees $\{t_{(z)} \mid z \in \Sigma_0^k\}$. The choice of k implies that there exist distinct words $z_1, z_2 \in \Sigma_0^k$ such that the sequences $w_{(z_1)}$ and $w_{(z_2)}$ of the form (3.10) are identical and A reaches the last node of the main paths of $t_{(z_1)}$ and $t_{(z_2)}$ in the same state.

The trees $t_{(z_i)}$ were constructed so that A visits exactly one of the leaves associated with each node of the main path before reaching the last node of the main path. Thus on the way up, at each node of the main path A visits exactly the leaves that were left unvisited on its way down. Since the main paths of $t_{(z_1)}$ and $t_{(z_2)}$ are identical and on its way down A visited the same sequence of leaves in $t_{(z_1)}$ and $t_{(z_2)}$, it follows that A has to accept also the

tree t obtained from $t_{(z_1)}$ by replacing the unvisited leaf at each level by the corresponding leaf symbol of $t_{(z_2)}$. However, $t \notin \mathcal{L}(\Sigma)$ because $z_1 \neq z_2$. ■

It is easy to see that the tree language $L(\Sigma)$ from the proof of Theorem 3.5 can be accepted by a nondeterministic one-visit caterpillar that on an input tree t nondeterministically checks that for each node u on $mp(t)$ the two leaves associated with u are identical and only after this makes a down-move to the next node on $mp(t)$. A similar strategy can be used by a deterministic 2-visit automaton that on a node u of $mp(t)$ visits all children of u, determines that the two leaves are identical, and after that continues to the node u' of $mp(t)$ that is a child of u. (Thus the automaton enters u' two times.)

Corollary 3.6 *The family* $\mathcal{L}(1\text{-CAT})$ *is strictly included in* $\mathcal{L}(2\text{-CAT})$.

4 Conclusion

The main open problem remaining is whether there exist regular tree languages that do not belong to $\mathcal{L}(NCAT)$. As we have argued above, we believe the conjecture from [3] to be true but finding a proof for it may be hard. Also we do not know whether the inclusion $\mathcal{L}(CAT) \subseteq \mathcal{L}(NCAT)$ is strict.

We have seen that two-visit caterpillars are strictly more powerful than one-visit caterpillars. It is not known whether any of the other inclusions in the k-visit hierarchy are strict. The proof of the strict inclusion $\mathcal{L}(1\text{-CAT}) \subset \mathcal{L}(2\text{-CAT})$ relied on a fooling technique where we were able to modify a part of the tree that the automaton has not yet visited, so similar arguments could not, at least not directly, be used to separate higher levels of the hierarchy.

Concerning closure properties of the caterpillar tree languages, it is not known whether $\mathcal{L}(NCAT)$ is closed under complementation [3]. In fact, even the closure of $\mathcal{L}(CAT)$ under complementation remains open. We have seen that given an arbitrary deterministic caterpillar A we can determine whether or not A admits an infinite loop. In the negative case, clearly also the complement of $L(A)$ is in $\mathcal{L}(CAT)$. However, we do not know how to construct a caterpillar accepting the complement of $L(A)$ in case A may reject some inputs by entering an infinite loop. Note that $\mathcal{L}(k\text{-CAT})$, $k \geq 1$, is closed under complementation.

A further open problem is whether given $A \in CAT$ we can determine in polynomial time whether or not A is a one-visit automaton.

References

[1] J. Berstel and L. Boasson, XML grammars. *Mathematical Foundations of Computer Science* 2000, Bratislava, Lect. Notes Comput. Sci. **1893**, pp. 182–191.

[2] J. Berstel and L. Boasson, Formal properties of XML grammars and languages. *Acta Informatica* **38** (2002) 649–671.

[3] A. Brüggemann-Klein and D. Wood, Caterpillars: A context-specification technique. *Mark-up Languages: Theory & Practice* **2** (2000) 81–106.

[4] A. Brüggemann-Klein and D. Wood, The regularity of two-way nondeterministic tree automata languages. *International Journal of Foundations of Computer Science* **13** (2002) 67–81.

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

[6] F. Gécseg and M. Steinby, *Tree Languages.* In: Handbook of Formal Languages, Vol. 3, G. Rozenberg and A. Salomaa (Eds.), Springer-Verlag, 1997, pp. 1–68.

[7] T. Kamimura: Tree automata and attribute grammars. *Information and Control* **57** (1983) 1–20.

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

[9] G. Slutzki: Alternating tree automata. *Theoretical Computer Science* **41** (1985) 305–318.

[10] D. Wood, Standard generalized markup language: Mathematical and philosophical issues. In: *Computer Science Today,* J. van Leeuwen (Ed.), Lect. Notes Comput. Sci. **1000**, Springer-Verlag 1995, pp. 344–365.