Tree Automata and XPath on Compressed Trees

Markus Lohrey[1] and Sebastian Maneth[2]

[1] FMI, University of Stuttgaert, Germany
lohrey@informatik.uni-stuttgart.de
[2] Faculté I & C, EPFL, Switzerland
sebastian.maneth@epfl.ch

Abstract. The complexity of various membership problems for tree automata on compressed trees is analyzed. Two compressed representations are considered: dags, which allow to share identical subtrees in a tree, and straight-line context-free tree grammars, which moreover allow to share identical intermediate parts of a tree. Several completeness results for the classes NL, P, and PSPACE are obtained. Finally, the complexity of the XPath evaluation problem on trees that are compressed via straight-line context-free tree grammars is investigated.

1 Introduction

During the last decade, the massive increase in the volume of data has motivated the investigation of algorithms on *compressed data*, like for instance compressed strings, trees, or pictures. The general goal is to develop algorithms that directly work on compressed data without prior decompression. Considerable amount of work has been done concerning algorithms on compressed strings, see e.g. [1, 2]. In this paper we investigate the computational complexity of algorithmic problems on *compressed trees*. Trees serve as a fundamental data structure in many fields of computer science, e.g. term rewriting, model checking, XML, etc. In fact, in each of these domains, compressed trees in form of *dags* (directed acyclic graphs), which allow to share identical subtrees in a tree, are used as a key for obtaining more efficient algorithms, see for instance [3] (term graph rewriting), [4] (model checking with BDDs), and [5, 6] (querying compressed XML documents). Recently, *straight-line context-free tree grammars* (SL cf tree grammars) were proposed as another compressed representation of trees in the context of XML [7]. Whereas a dag can be seen as a *regular* tree grammar [8] that generates exactly one tree, an SL cf tree grammar is a *context-free tree grammar* [8] that generates exactly one tree. SL cf tree grammars allow to share identical intermediate parts in a tree. This results in better compression rates in comparison to dags: in the theoretical optimum, SL cf tree grammars lead to doubly exponential compression rates, whereas dags only allow singly exponential compression rates. In [9], a practical algorithm (BPLEX) for generating a small SL cf tree grammar that produces a given input tree is presented. Experiments with existing XML benchmark data show that BPLEX results in significantly better compression rates than dag-based compression algorithms.

In Section 3 we study the problem of evaluating compressed trees via *tree automata* [8, 10]. Tree automata play a fundamental role in many applications where trees have to be processed in a systematic way. In the context of XML, for instance, tree automata are used to type check documents against an XML type [11, 12]. These applications motivate the investigation of general decision problems for tree automata like emptiness, equivalence, and intersection nonemptiness. Several complexity results are known for these problems, see e.g. [8]. Membership problems for tree automata were investigated in [13] for ranked trees (see Table 1 for the results of [13]) and [14] for unranked trees from the perspective of computational complexity. Here we extend this line of research by investigating the computational complexity of membership problems for various classes of tree automata on compressed trees (dags and SL cf tree grammars). For deterministic/nondeterministic top-down/bottom-up tree automata we analyze the fixed membership problem (where the tree automaton is not part of the input) as well as the uniform membership problem (where the tree automaton is also part of the input). Moreover, we consider subclasses of SL cf tree grammars that allow more efficient algorithms for evaluating tree automata. In particular, linearity and the restriction that for some constant k, every production of the SL cf tree grammar contains at most k parameters (variables) lead to better complexity bounds. For all cases, we present upper and lower bounds which vary from NL (nondeterministic logspace) to PSPACE (polynomial space). Our results are collected in Table 1. We also briefly consider the parameterized complexity [15] of membership problems for tree automata.

In Section 4 we consider the problem of evaluating core XPath expressions over compressed trees. XPath is a widely used language for selecting nodes in XML documents and is the core of many modern XML technologies. The query problem for XPath asks whether a given node in a given (unranked) tree is selected by a given XPath expression. For uncompressed trees, the complexity of this problem is intensively studied in [16, 17]. For input trees that are represented as dags, XPath evaluation was investigated in [5, 6]. In [6] it was shown that the evaluation problem for core XPath (the navigational part of XPath) over dag-compressed trees is PSPACE-complete. Here, we extend this result to linear SL cf tree grammars (Theorem 9). This is remarkable, since linear SL cf tree grammars lead to (provably) better compression rates than dags, which is also confirmed by our experimental results for the BPLEX-algorithm (which produces linear SL cf tree grammars) from [9].

Proofs that are omitted in the main part of this paper will appear in the full version.

2 Preliminaries

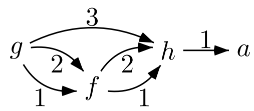
For background in complexity theory see [18]. The set of all finite strings over a (not necessarily finite) alphabet Σ is Σ^*. The empty string is ε. The length of a string u is $|u|$. We write $u \preceq v$ for $u, v \in \Sigma^*$ if u is a prefix of v. The reflexive and transitive closure of a binary relation $\rightarrow$ is denoted by $\xrightarrow{*}$.

Trees, dags, and SL cf tree grammars A *ranked alphabet* is a pair $(\mathcal{F}, \mathrm{arity})$, where $\mathcal{F}$ is a finite set of function symbols and $\mathrm{arity} : \mathcal{F} \to \mathbb{N}$ assigns to each $\alpha \in \mathcal{F}$ its arity (or rank). Let $\mathcal{F}_i = \{\alpha \in \mathcal{F} \mid \mathrm{arity}(\alpha) = i\}$. Function symbols in $\mathcal{F}_0$ are called *constants*. In examples we use symbols $a \in \mathcal{F}_0, h \in \mathcal{F}_1$, and $f \in \mathcal{F}_2$. Mostly we omit the function arity in the description of a ranked alphabet. An *$\mathcal{F}$-labeled tree* t (or *ground term* over $\mathcal{F}$) is a pair $t = (\mathrm{dom}_t, \lambda_t)$, where (i) $\mathrm{dom}_t \subseteq \mathbb{N}^*$ is finite, (ii)

$\lambda_t : \mathrm{dom}_t \to \mathcal{F}$, (iii) if $v \preceq w \in \mathrm{dom}_t$, then also $v \in \mathrm{dom}_t$, and (iv) if $v \in \mathrm{dom}_t$ and $\lambda_t(v) \in \mathcal{F}_n$, then $vi \in \mathrm{dom}_t$ if and only if $1 \leq i \leq n$. Note that the edge relation of the tree t can be defined as $\{(v, vi) \in \mathrm{dom}_t \times \mathrm{dom}_t \mid v \in \mathbb{N}^*, i \in \mathbb{N}\}$. The size of t is $|t| = |\mathrm{dom}_t|$. With an $\mathcal{F}$-labeled tree t we associate a term in the usual way: If $\lambda_t(\varepsilon) = \alpha \in \mathcal{F}_i$, then this term is $\alpha(t_1, \ldots, t_i)$, where t_j is the term that corresponds to the subtree of t rooted at the node $j \in \mathbb{N}$. The set of all $\mathcal{F}$-labeled trees is $T(\mathcal{F})$. Let us fix a countable set $\mathcal{X}$ of variables. The set of all $\mathcal{F}$-labeled trees with variables from $\mathcal{X}$ is $T(\mathcal{F}, \mathcal{X})$. Formally, we consider variables as new constants and define $T(\mathcal{F}, \mathcal{X}) = T(\mathcal{F} \cup \mathcal{X})$. A tree $t \in T(\mathcal{F}, \mathcal{X})$ is *linear*, if every variable $x \in \mathcal{X}$ occurs at most once in t. A *term rewriting system*, briefly TRS, over a ranked alphabet $\mathcal{F}$ is a finite set $\mathcal{R} \subseteq (T(\mathcal{F}, \mathcal{X}) \setminus \mathcal{X}) \times T(\mathcal{F}, \mathcal{X})$ such that for all $(s, t) \in \mathcal{R}$, every variable that occurs in t also occurs in s. The *one-step rewrite relation* $\to_{\mathcal{R}}$ over $T(\mathcal{F}, \mathcal{X})$ is defined as usual, see for instance [19].

Dags (directed acyclic graphs) are a popular compressed representation of trees that allows to share identical subtrees. An $\mathcal{F}$-labeled *dag* is a triple $D = (V_D, \lambda_D, E_D)$ where (i) V_D is a finite set of nodes, (ii) $\lambda_D : V_D \to \mathcal{F}$ labels each node with a symbol from $\mathcal{F}$, (iii) $E_D \subseteq V_D \times \mathbb{N} \times V_D$ (i.e. edges are directed and labeled with natural numbers), (iv) every $v \in V_D$ contains precisely one i-labeled outgoing edge for every $1 \leq i \leq \mathrm{arity}(\lambda_D(v))$, and (v) (V_D, E_D) is acyclic and contains precisely one node $\mathrm{root}_D \in V_D$ without incoming edges. The size of D is $|D| = |V_D|$. A *root-path* in D is a path $v_1, i_1, v_2, i_2 \cdots, v_n$ in the graph (V_D, E_D), i.e., $v_k \in V_D$ $(1 \leq k \leq n)$ and $(v_k, i_k, v_{k+1}) \in E_D$ $(1 \leq k < n)$ that moreover starts in the root node, i.e., $v_1 = \mathrm{root}_D$. Such a path can be identified with the label-sequence $i_1 i_2 \cdots i_{n-1} \in \mathbb{N}^*$. An $\mathcal{F}$-labeled dag D over $\mathcal{F}$ can be unfolded into an $\mathcal{F}$-labeled tree $\mathrm{eval}(D)$: $\mathrm{dom}_{\mathrm{eval}(D)}$ is the set of all root-paths in D (viewed as a subset of $\mathbb{N}^*$), and if the root-path $p \in \mathbb{N}^*$ ends in the node $v \in V_D$, then we set $\lambda_{\mathrm{eval}(D)}(p) = \lambda_D(v)$. Clearly the size of $\mathrm{eval}(D)$ is bounded exponentially in $|D|$.

Example 1. For the dag D on the right we have $\mathrm{eval}(D) = g(f(h(a), h(a)), f(h(a), h(a)), h(a))$. Moreover, the size of D is 4. We have $\mathrm{dom}_{\mathrm{eval}(D)} = \{\varepsilon, 1, 2, 3, 11, 12, 21, 22, 31, 111, 121, 211, 221\}$.



Recently, a compressed representation of trees, which generalizes dags, was introduced: *straight-line context-free tree grammars* (*SL cf tree grammars*) [7]. An SL cf tree grammar is a tuple $G = (\mathcal{F}, N, S, P)$, where (i) $N \cup \mathcal{F}$ is a ranked alphabet, (ii) N is the set of nonterminals, (iii) $\mathcal{F}$ is the set of terminals, (iv) $S \in N$ is the start nonterminal and has rank 0, (v) P (the set of productions) is a TRS over $N \cup \mathcal{F}$ that contains for every $A \in N$ exactly one rule of the form $A(x_1, \ldots, x_n) \to t_A$, where $n = \mathrm{arity}(A)$ and $x_1, \ldots, x_n$ are pairwise different variables, and (vi) the relation $\{(A, B) \in N \times N \mid B \text{ occurs in } t_A\}$ is acyclic. These conditions ensure that for every $A \in N$ of rank n there is a unique tree $\mathrm{eval}_G(A)(x_1, \ldots, x_n) \in T(\mathcal{F}, \{x_1, \ldots, x_n\})$ with $A(x_1, \ldots, x_n) \xrightarrow{*}_P \mathrm{eval}_G(A)(x_1, \ldots, x_n)$. Let $\mathrm{eval}(G) = \mathrm{eval}_G(S) \in T(\mathcal{F})$. Thus, an SL cf tree grammar is a context free tree grammar [8] that generates exactly one tree. Alternatively, an SL cf tree grammar is a *recursive program scheme* [20] that generates a finite tree. The size of G is $|G| = \sum_{A \in N} |t_A|$. We say that G is an SL cf tree grammar *with k parameters* ($k \geq 0$) if $\mathrm{arity}(A) \leq k$ for every $A \in N$. The SL cf tree grammar G is *linear* if for every production $A(x_1, \ldots, x_n) \to t_A$ in P the tree t_A is linear.

SL cf tree grammars generalize string generating straight-line programs [2] in a natural way from strings to trees. The following example shows that SL cf tree grammars may lead to doubly exponential compression rates; thus, they can be exponentially more succinct than dags: Let the (non-linear) SL cf tree grammar G_n consist of the following productions: $S \to A_0(a)$, $A_i(x) \to A_{i+1}(A_{i+1}(x))$ for $0 \leq i < n$, and $A_n(x) \to f(x, x)$. Then $\mathrm{eval}(G_n)$ is a complete binary tree of height 2^n. Thus, $|\mathrm{eval}(G_n)| \in O(2^{2^n})$. Note that G_n has only one parameter. On the other hand, it is easy to prove by induction over the number of productions that *linear* SL cf tree grammars can only achieve exponential compression rates. But linear SL cf tree grammars are still more succinct than dags: The tree $h(h(\cdots h(a) \cdots))$ with 2^n many occurrences of h can be generated by a linear SL cf tree grammar of size $\mathcal{O}(n)$, which is not possible with dags.

An SL cf tree grammar $G = (\mathcal{F}, N, S, P)$ with 0 parameters (i.e., $\mathrm{arity}(A) = 0$ for every nonterminal $A \in N$) can be easily transformed in logspace into an $\mathcal{F}$-labeled dag that generates the same tree: we take the disjoint union of all right-hand sides of productions from P, where the root of the right-hand side for the nonterminal A gets the additional label A. Then we merge for every nonterminal A all nodes with label A. Note that since $\mathrm{arity}(A) = 0$ for every $A \in N$, nonterminals can only occur as leafs in right-hand sides of G. Thus, this merging process results in a dag. For instance, the SL cf tree grammar with the productions $S \to g(A, A, B), A \to f(B, B), B \to h(a)$ corresponds to the dag from Example 1. Vice versa, from an $\mathcal{F}$-labeled dag we can construct in logspace an equivalent SL cf tree grammar with 0 parameters by taking the nodes of the dag as nonterminals. Thus, dags can be seen as special SL cf tree grammars. This justifies our choice to denote with eval both the evaluation function for dags and unrestricted SL cf tree grammars.

Tree automata A (nondeterministic) *top-down tree automaton*, briefly TDTA, is a tuple $\mathcal{A} = (Q, \mathcal{F}, q_0, \mathcal{R})$, where Q is a finite set of states, $Q \cup \mathcal{F}$ is a ranked alphabet with $\mathrm{arity}(q) = 1$ for all $q \in Q$, $q_0 \in Q$ is the initial state, and $\mathcal{R}$ is a TRS such that all rules have the form $q(\alpha(x_1, \ldots, x_n)) \to \alpha(q_1(x_1), \ldots, q_n(x_n))$, where $q, q_1, \ldots, q_n \in Q$, $x_1, \ldots, x_n$ are pairwise different variables, and $\alpha \in \mathcal{F}$ has rank n. $\mathcal{A}$ is a *deterministic TDTA* if no two rules in $\mathcal{R}$ have the same left-hand side. The tree language that is accepted by a TDTA $\mathcal{A}$ is $T(\mathcal{A}) = \{t \in T(\mathcal{F}) \mid q_0(t) \xrightarrow{*}_{\mathcal{R}} t\}$. A (nondeterministic) *bottom-up tree automaton*, briefly BUTA, is a tuple $\mathcal{A} = (Q, \mathcal{F}, Q_f, \mathcal{R})$, where Q and $\mathcal{F}$ are as above, $Q_f \subseteq Q$ is the set of final states, and $\mathcal{R}$ is a TRS such that all rules have the form $\alpha(q_1(x_1), \ldots, q_n(x_n)) \to q(\alpha(x_1, \ldots, x_n))$, where $q, q_1, \ldots, q_n \in Q$, $x_1, \ldots, x_n$ are pairwise different variables, and $\alpha \in \mathcal{F}$ has rank n. $\mathcal{A}$ is a *deterministic BUTA* if no two rules in $\mathcal{R}$ have the same left-hand side. The tree language that is accepted by a BUTA $\mathcal{A}$ is $T(\mathcal{A}) = \{t \in T(\mathcal{F}) \mid \exists q \in Q_f : t \xrightarrow{*}_{\mathcal{R}} q(t)\}$. It is straight-forward to transform a nondeterministic BUTA into an equivalent nondeterministic TDTA and vice versa, and a logspace transducer is able to to do these transformations. Thus, in the following we do not distinguish between nondeterministic BUTA and nondeterministic TDTA, and we call them simply tree automata (TA). A subset of

Table 1. Complexity results for (uniform) membership problems

<table>
<tr><th></th><th></th><th>det. TDTA</th><th>det. BUTA</th><th>TA</th></tr>
<tr><td rowspan="2">uncompressed trees [13]</td><td>fixed</td><td colspan="3">NC^1-complete</td></tr>
<tr><td>uniform</td><td>L-complete</td><td>LOGDCFL,
L-hard</td><td>LOGCFL-
complete</td></tr>
<tr><td rowspan="2">dags</td><td>fixed</td><td rowspan="2">NL-complete</td><td rowspan="2" colspan="2">P-complete</td></tr>
<tr><td>uniform</td></tr>
<tr><td rowspan="2">lin. SL + fixed number para.</td><td>fixed</td><td rowspan="2" colspan="3">P-complete</td></tr>
<tr><td>uniform</td></tr>
<tr><td rowspan="2">SL + fixed number para.</td><td>fixed</td><td rowspan="2" colspan="2">P-complete</td><td></td></tr>
<tr><td>uniform</td><td>PSPACE-
complete</td></tr>
<tr><td rowspan="2">unrestricted SL</td><td>fixed</td><td rowspan="2">P-complete</td><td rowspan="2" colspan="2">PSPACE-complete</td></tr>
<tr><td>uniform</td></tr>
</table>

$T(\mathcal{F})$ is *recognizable* if it is accepted by a TA. Using a powerset construction, every recognizable tree language can be also accepted by a deterministic BUTA, but this involves an exponential blowup in the number of states. For deterministic TDTA the situation is different; they only recognize a proper subclass of the recognizable tree languages. The size $|\mathcal{A}|$ of a TA is the sum of the sizes of all left and right hand sides of rules. Let $\mathcal{G}$ be a class of SL cf tree grammars (e.g., the class of all dags). The *membership problem* for the fixed TA $\mathcal{A}$ and the class $\mathcal{G}$ is the following decision problem:

INPUT: $G \in \mathcal{G}$
QUESTION: Does $\mathrm{eval}(G) \in T(\mathcal{A})$ hold?

For a class $\mathcal{C}$ of tree automata, the *uniform membership problem* for $\mathcal{C}$ and the class $\mathcal{G}$ is the following decision problem:

INPUT: $G \in \mathcal{G}$ and $\mathcal{A} \in \mathcal{C}$
QUESTION: Does $\mathrm{eval}(G) \in T(\mathcal{A})$ hold?

The upper part of Table 1 collects the complexity results that were obtained in [13] for uncompressed trees. The statement that for instance the membership problem for TA is NC^1-complete means that for every fixed TA the membership problem is in NC^1 and that there exists a fixed TA for which the membership problem is NC^1-hard. More details on tree automata can be found in [8, 10].

3 Membership Problems for Dags and SL CF Tree Grammars

The time bounds in the following theorem are based on dynamic programming. Note that only the number k of parameters appears in the exponent. The idea of the proof is to run the tree automaton $\mathcal{A}$ bottom up on the right-hand sides of G's productions. For the parameters we have to assume at most n^k different possibilities of states of $\mathcal{A}$ which (a determinized simulation of) $\mathcal{A}$ maps to a state of $\mathcal{A}$.

Theorem 1. *For a given TA $\mathcal{A}$ with n states and a linear SL cf tree grammar G with k parameters we can check in time $\mathcal{O}(n^{k+1} \cdot |G| \cdot |\mathcal{A}|)$ whether* $\mathrm{eval}(G) \in T(\mathcal{A})$.

For a given deterministic BUTA $\mathcal{A}$ with n states and a given SL cf tree grammar with k parameters we can check in time $\mathcal{O}(n^k \cdot |G| \cdot |\mathcal{A}|)$ whether $\mathrm{eval}(G) \in T(\mathcal{A})$.

Recall that a dag can be seen as a (linear) SL cf tree grammar without parameters. Thus, Theorem 1 can be also applied to dags in order to obtain a polynomial time algorithm for the uniform membership problem for TA and dags. Using a straightforward reduction from the P-complete monotone circuit-value problem, we obtain:

Theorem 2. *There exists a fixed deterministic BUTA $\mathcal{A}$ such that the membership problem for $\mathcal{A}$ and dags is P-hard.*

Remark 1. By Theorem 1 and 2, the (uniform) membership problem for (deterministic) BUTA on dags is P-complete. This result may appear surprising when compared with a recent result from [21]: the membership problem for so called dag automata is NP-complete. But in contrast to our approach, a dag automaton operates directly on a dag, whereas we consider ordinary tree automata that run on the unfolded dag. This makes a crucial difference for the complexity of the membership problem.

By the next theorem, a deterministic TDTA can be evaluated on a dag in NL (nondeterministic logspace). The crucial fact is that a deterministic TDTA $\mathcal{A}$ accepts a tree t if and only if the path language of t (which is, roughly speaking, the set of all words labeling a maximal path in the tree t) is included in some regular string language L [10], where L is accepted by a finite automaton $\mathcal{B}$ that is logspace constructible from $\mathcal{A}$. Now we just guess a path in the input dag and simulate $\mathcal{B}$ on this path. The NL lower bound is obtained by a reduction from the graph accessibility problem for dags.

Theorem 3. *The uniform membership problem for deterministic TDTA and dags is in NL. Moreover, there exists a fixed deterministic TDTA such that the membership problem for $\mathcal{A}$ and dags is NL-hard.*

By combining the statements in Theorem 1–3 we obtain the results for dags in Table 1.

SL cf tree grammars allow higher compression rates than dags. This makes computational problems harder when input trees are represented via SL cf tree grammars. The following result reflects this phenomenon. The PSPACE lower bound can be shown by a reduction from QSAT (quantified boolean satisfiability), see e.g. [18].

Theorem 4. *The uniform membership problem for TA and SL cf tree grammars is in PSPACE. Moreover, there exists a fixed deterministic BUTA such that the membership problem for $\mathcal{A}$ and SL cf tree grammars is PSPACE-hard.*

Only for deterministic TDTA we obtain more efficient algorithms in the context of unrestricted SL cf tree grammars. The polynomial time upper bound in the next theorem is again based on the concept of the path language of a tree. For an SL cf tree grammar G, the path language of $\mathrm{eval}(G)$ can be generated by a small context-free string grammar. The lower bound follows from a result of [22] about string straight-line programs.

Theorem 5. *The uniform membership problem for deterministic TDTA and SL cf tree grammars is in P. Moreover, there is a fixed deterministic TDTA such that the membership problem for $\mathcal{A}$ and linear SL cf tree grammars with only one parameter is P-hard.*

From Theorem 1 and 5 (resp. Theorem 4 and 5) we obtain the complexity results for linear SL cf tree grammars with a fixed number of parameters (resp. unrestricted SL cf tree grammars) in Table 1, see lin. SL + fixed number para. (resp. unrestricted SL). The following result completes our characterization presented in Table 1.

Theorem 6. *The uniform membership problem for TA and (non-linear) SL cf tree grammars with only one parameter is PSPACE-hard.*

Proof. We prove the theorem by a reduction from QSAT [18]. Let us take a quantified boolean formula $\psi = Q_1 x_1 \cdots Q_n x_n\, \varphi$, where $Q_i \in \{\forall, \exists\}$ and φ is a boolean formula with variables from $\mathcal{X} = \{x_1, \ldots, x_n\}$. W.l.o.g. we may assume that in φ the negation operator $\neg$ only occurs directly in front of variables. Let $\bar{\mathcal{X}} = \{\neg x \mid x \in \mathcal{X}\}$. We define an SL cf tree grammar G as follows: The set of terminals contains the binary function symbol f, a unary function symbol t_i for every $x_i \in \mathcal{X}$, and a constant a. The set of nonterminals contains the start nonterminal S, and for every subformula α of ψ it contains a nonterminal A_α of arity 1. The productions of G are:

$$
\begin{array}{ll}
S \to A_\psi(a) & A_\alpha(y) \to f(A_\beta(t_i(y)), A_\beta(y)) \text{ if } \alpha \in \{\forall x_i \beta, \exists x_i \beta\} \\
A_\alpha(y) \to y \text{ if } \alpha \in \mathcal{X} \cup \bar{\mathcal{X}} & A_\alpha(y) \to f(A_\beta(y), A_\gamma(y)) \quad \text{ if } \alpha \in \{\beta \wedge \gamma, \beta \vee \gamma\}
\end{array}
$$

An occurrence of the symbol t_i on a path in the tree $\mathrm{eval}(G)$ indicates that the variable x_i is set to true. Note that from a nonterminal A_α, where α begins with a quantification $\exists x_i$ or $\forall x_i$ we first generate a branching node (labeled with the binary symbol f). Moreover, the left branch gets in addition the unary symbol t_i, which indicates that x_i is set to true. The absence of t_i in the right branch indicates that x_i is set to false.

We define a nondeterministic TDTA $\mathcal{A}$ as follows: The state set of $\mathcal{A}$ contains all subformulas of ψ plus an additional state q. The initial state of $\mathcal{A}$ is the whole formula ψ. The set $\mathcal{R}$ of transition rules of $\mathcal{A}$ consists of the following rules:

$$
\begin{array}{rll}
q(f(y,z)) \to & f(q(y), q(z)) & \\
q(t_i(y)) \to & t_i(q(y)) & \text{for all } i \\
q(a) \to & a & \\
\alpha(f(y,z)) \to & f(\beta(y), q(z)) & \text{if } \alpha = \exists x_i \beta \text{ for some } i \\
\alpha(f(y,z)) \to & f(q(y), \beta(z)) & \text{if } \alpha = \exists x_i \beta \text{ for some } i \\
\alpha(f(y,z)) \to & f(\beta(y), \beta(z)) & \text{if } \alpha = \forall x_i \beta \text{ for some } i \\
\alpha(f(y,z)) \to & f(\beta(y), q(z)) & \text{if } \alpha = \beta \vee \gamma \text{ for some } \gamma \\
\alpha(f(y,z)) \to & f(q(y), \gamma(z)) & \text{if } \alpha = \beta \vee \gamma \text{ for some } \beta \\
\alpha(f(y,z)) \to & f(\beta(y), \gamma(z)) & \text{if } \alpha = \beta \wedge \gamma \\
\alpha(t_i(y)) \to & t_i(\alpha(y)) & \text{if } \alpha \in (\mathcal{X} \cup \bar{\mathcal{X}}) \setminus \{x_i, \neg x_i\} \\
\alpha(t_i(y)) \to & t_i(q(y)) & \text{if } \alpha = x_i \\
\alpha(a) \to & a & \text{if } \alpha \in \bar{\mathcal{X}}
\end{array}
$$

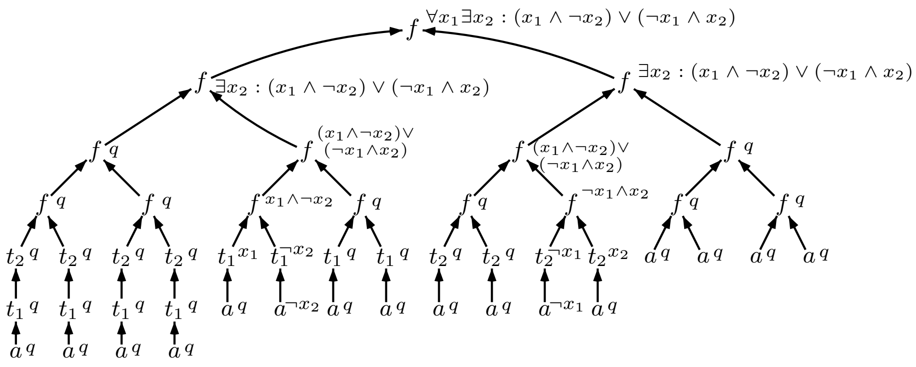


Fig. 1.

Figure 1 shows the tree $\mathrm{eval}(G)$ for the true quantified boolean formula $\forall x_1 \exists x_2 : (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$, where in addition every node is labeled with a state of the automaton $\mathcal{A}$ such that the overall labeling is an accepting run.

By the first three rules for state q, $q(t) \xrightarrow{*}_{\mathcal{R}} t$ for every ground tree t. Thus, if we reach the state q, then the corresponding subtree is accepted. If the current state α is an existential subformula $\exists x_i \beta$, then we guess nondeterministically one of the two subtrees of the current f-labeled node (i.e., we choose an assignment for x_i) and verify β in that subtree. The other subtree is accepted by sending q to that subtree. Similarly, if the current state α is a universal subformula $\forall x_i \beta$, then we verify β in both subtrees, i.e., for both assignments for x_i. The rules for $\alpha = \beta \vee \gamma$ and $\alpha = \beta \wedge \gamma$ can be interpreted similarly. Note that by construction of G and $\mathcal{A}$, if the current state α is of the form $\exists x_i \beta$, $\forall x_i \beta$, $\beta \vee \gamma$, or $\beta \wedge \gamma$, then the current tree node in $\mathrm{eval}(G)$ is an f-labeled node. On the other hand, if the current state is from $\mathcal{X} \cup \bar{\mathcal{X}}$, then the current tree node in $\mathrm{eval}(G)$ is labeled with a symbol t_j or the constant a. If the current state is a variable x_i, then we search for the symbol t_i in the chain of t_j-labeled nodes below the current node. We accept by going into the state q as soon as we find t_i: $x_i(t_i(y)) \to t_i(q(y))$ If we do not find t_i and end up in the constant a, then we block; note that there is no rule of form $x_i(a) \to a$. On the other hand, if the current state is a negated variable $\neg x_i$, then we verify that there is no t_i in the chain of t_j-labeled nodes below the current node. Thus, we block as soon as we find t_i; note that there is no rule with left-hand side $\neg x_i(t_i(y))$. On the other hand, if we finally reach the constant a in state $\neg x_i$, then we accept via the rule $\neg x_i(a) \to a$. From the previous discussion, it is not hard to see that the formula ψ is true if and only if $\mathrm{eval}(G) \in L(\mathcal{A})$. $\square$

From Theorem 1 and Theorems 4–6 we obtain the results for SL cf tree grammars with a fixed number of parameters in Table 1.

We end this sections with two results concerning the parameterized complexity of membership problems for tree automata. *Parameterized complexity* [15] is a branch of complexity theory with the goal to understand which input parts of a hard (e.g. NP-hard) problem are responsible for the combinatorial explosion. A *parameterized problem* is a decision problem where the input is a pair $(k, x) \in \mathbb{N} \times \Sigma^*$. The first input component k is called the *input parameter* (it may also consist of several natural numbers). A typical example of a parameterized problem is the parameterized version of the clique problem, where the input is a pair (k, G), G is an undirected graph, and it is asked whether G has a clique of size k. A parameterized problem (with input (k, x)) is in the class FPT (fixed parameter tractable), if the problem can be solved in time $f(k) \cdot |x|^c$. Here c is a fixed constant and f is an arbitrary (e.g., exponential) computable function on $\mathbb{N}$. This means that the non-polynomial part of the algorithm is restricted to the parameter k.

Theorem 7. *The following parameterized problem is in FPT:*

INPUT: An SL cf tree grammar G with k parameters and a TA $\mathcal{A}$ with n states.
INPUT PARAMETER: (k, n)
QUESTION: $\mathrm{eval}(G) \in T(\mathcal{A})$*?*

Proof. We first transform $\mathcal{A}$ into a deterministic BUTA with at most 2^n states. Then we apply Theorem 1 which gives us a running time of $2^{kn} \cdot |G| \cdot |\mathcal{A}|$. □

In recent years, a structural theory of parameterized complexity with the aim of showing that certain problems are unlikely to belong to FPT was developed. Underlying this theory is the notion of parameterized reductions [15]: A parameterized reduction from a parameterized problem A (with input $(k, x) \in \mathbb{N} \times \Sigma^*$) to a parameterized problem B (with input $(\ell, y) \in \mathbb{N} \times \Gamma^*$) is a mapping $f : \mathbb{N} \times \Sigma^* \to \mathbb{N} \times \Gamma^*$ such that: (i) for all $(k, x) \in \mathbb{N} \times \Sigma^*$, $(k, x) \in A$ if and only if $f(k, x) \in B$, (ii) $f(k, x)$ is computable in time $g(k) \cdot |x|^c$ for some computable function g and some constant c, and (iii) for some computable function h, if $f(k, x) = (\ell, y)$, then $\ell \leq h(k)$. A parameterized problem A is fpt-reducible to a parameterized problem B if there exists a parameterized reduction from A to B. One of the classes in the upper part of the parameterized complexity spectrum is the class AW[P]. For the purpose of this paper it is not necessary to present the quite technical definition of AW[P]. Roughly speaking, AW[P] results from taking the closure (w.r.t. fpt-reducibility) of a parameterized version of the PSPACE-complete QSAT problem. Problems that are AW[P]-hard are very unlikely to be in FPT.

Theorem 8. *The following problem is AW[P]-hard w.r.t. fpt-reducibility:*

INPUT: A deterministic BUTA $\mathcal{A}$ and an SL cf tree grammar G with k parameters
INPUT PARAMETER: k
QUESTION: $\mathrm{eval}(G) \in T(\mathcal{A})$*?*

The theorem can be shown by a parameterized reduction from the following problem pFOMC (parameterized first-order model-checking), which is AW[P]-hard w.r.t. fpt-reducibility [23]:

INPUT: A directed graph $H = (V, E)$ and a sentence ϕ of first-order logic (built up from the atomic formulas $x = y$ and $E(x, y)$ (for variables x and y) using boolean connectives and quantification over nodes of H).
INPUT PARAMETER: The number of different variables that are used in ϕ
QUESTION: Is ϕ true in the graph H?

4 XPath Evaluation

In this section, we consider XML-trees that are compressed via SL cf tree grammars and study the node selecting language XPath over such trees. For more background on XPath see [16, 17]. We restrict our attention to linear SL cf tree grammars. Skeletons of XML documents are usually modeled as *rooted unranked labeled trees*. Analogously to Section 2, an unranked tree with labels from an (unranked) alphabet Σ can be defined as a pair $t = (\mathrm{dom}_t, \lambda_t)$, where (i) $\mathrm{dom}_t \subseteq \mathbb{N}^*$ is finite, (ii) $\lambda_t : \mathrm{dom}_t \to \mathcal{F}$, (iii) if $v \preceq w \in \mathrm{dom}_t$, then also $v \in \mathrm{dom}_t$, and (iv) if $vi \in \mathrm{dom}_t$ then also $vj \in \mathrm{dom}_t$ for every $1 \leq j \leq i$. For the purpose of this section, it is more suitable to view such an unranked tree $t = (\mathrm{dom}_t, \lambda_t)$ as a relational structure $t = (\mathrm{dom}_t, \text{child}, \text{next-sibling}, (Q_a)_{a\in\Sigma})$, where $Q_a = \lambda_t^{-1}(a) \subseteq \mathrm{dom}_t$, child $= \{(v, vi) \in \mathrm{dom}_t \times \mathrm{dom}_t \mid v \in \mathbb{N}^*, i \in \mathbb{N}\}$, and next-sibling $= \{(vi, v(i+1)) \in \mathrm{dom}_t \times \mathrm{dom}_t \mid v \in \mathbb{N}^*, i \in \mathbb{N}\}$. Thus, child$(u, v)$ is the child-relation in t and next-sibling(u, v) if and only if v is the right sibling of u. From the basic tree relations child and next-sibling further tree relations that are called *XPath-axes* can be defined. For instance let descendant := child* (the reflexive and transitive closure of child) and following-sibling := next-sibling*. For the definition of the other XPath axes see for instance [16]. In the following we consider the four XPath axes child, descendant, next-sibling, and following-sibling; handling of other axes is straightforward and needs no further ideas.

The node selection language *core XPath* [16] can be seen as the tree navigational core of XPath. Its syntax is given by the following EBNF; here, χ is an XPath-axis and $a \in \Sigma \cup \{*\}$ (where $*$ is a new symbol):

$$\begin{aligned}
\text{corexpath} &::= \text{locationpath} \;\mid\; /\,\text{locationpath}\\
\text{locationpath} &::= \text{locationstep}\,(/\,\text{locationstep})^*\\
\text{locationstep} &::= \chi :: a \;\mid\; \chi :: a\,[\text{pred}]\\
\text{pred} &::= (\text{pred and pred}) \;\mid\; (\text{pred or pred}) \;\mid\; \text{not(pred)} \;\mid\; \text{locationpath}
\end{aligned}$$

Let Q_* be the unary predicate that is true for every node of a tree t. We define the semantics of core XPath by translating a given tree $t = (\mathrm{dom}_t, \text{child}, \text{next-sibling}, (Q_a)_{a\in\Sigma})$ and a given expression $\pi \in \mathcal{L}(\text{corexpath})$ (resp. $e \in \mathcal{L}(\text{pred})$) into a binary relation $\mathcal{S}[\pi, t] \subseteq \mathrm{dom}_t \times \mathrm{dom}_t$ (resp. a unary relation $\mathcal{E}[e, t] \subseteq \mathrm{dom}_t$). Let $\pi, \pi_1, \pi_2 \in \mathcal{L}(\text{locationpath})$, $e, e_1, e_2 \in \mathcal{L}(\text{pred})$, and let χ be an XPath axes (recall that ε is the root of a tree).

$$\begin{aligned}
\mathcal{S}[\chi :: a[e], t] &:= \{(x, y) \in \mathrm{dom}_t \times \mathrm{dom}_t \mid (x, y) \in \chi, y \in Q_a, y \in \mathcal{E}[e, t]\}\\
\mathcal{S}[/\pi, t] &:= \mathrm{dom}_t \times \{x \in \mathrm{dom}_t \mid (\varepsilon, x) \in \mathcal{S}[\pi, t]\}\\
\mathcal{S}[\pi_1/\pi_2, t] &:= \{(x, y) \in \mathrm{dom}_t \times \mathrm{dom}_t \mid \exists z : (x, z) \in \mathcal{S}[\pi_1, t], (z, y) \in \mathcal{S}[\pi_2, t]\}\\
\mathcal{E}[e_1 \text{ and } e_2, t] &:= \mathcal{E}[e_1, t] \cap \mathcal{E}[e_2, t]\\
\mathcal{E}[e_1 \text{ or } e_2, t] &:= \mathcal{E}[e_1, t] \cup \mathcal{E}[e_2, t]\\
\mathcal{E}[\text{not}(e), t] &:= \mathrm{dom}_t \setminus \mathcal{E}[e, t]\\
\mathcal{E}[\pi, t] &:= \{x \in \mathrm{dom}_t \mid \exists y : (x, y) \in \mathcal{S}[\pi, t]\}
\end{aligned}$$

Recall that by definition SL cf tree grammars generate ranked trees. In order to generate XML skeletons, i.e., unranked trees, with SL cf tree grammars, we encode unranked trees by binary trees (and hence ranked trees) using a standard encoding: For an unranked tree $t = (\mathrm{dom}_t, \text{child}, \text{next-sibling}, (Q_a)_{a\in\Sigma})$ define the binary encoding $\mathrm{bin}(t) = (\mathrm{dom}_t, \text{child1}, \text{child2}, (Q_a)_{a\in\Sigma})$, where (i) $(u, v) \in$ child1 if and only if $(u, v) \in$ child and there does not exist $w \in \mathrm{dom}_t$ with $(w, v) \in$ next-sibling (i.e., v is the left-most child of u), and (ii) child2 $=$ next-sibling. Note that t and $\mathrm{bin}(t)$ have the same set of nodes. The following theorem is our main result in this section. PSPACE-hardness follows from the corresponding result for dags [6].

Theorem 9. *The following problem is PSPACE-complete:*

INPUT: A linear SL cf tree grammar G generating a binary tree with $\mathrm{eval}(G) = \mathrm{bin}(t)$ *for some (unique) unranked tree t, two nodes u, v of* $\mathrm{eval}(G)$*, and a core XPath expression* $\pi \in \mathcal{L}(\text{corexpath})$.

QUESTION: $(u, v) \in \mathcal{S}[\pi, t]$ *?*

For the proof of the PSPACE upper bound in Theorem 9 we first translate a given XPath expression into a first-order formula that uses the XPath axes as atomic predicates. We then show that such a first-order formula can be evaluated on $\mathrm{eval}(G)$ for a given linear SL cf tree grammar by an alternating Turing machine [18] that works in polynomial time with respect to the size of the formula and the size of the grammar. For this it is crucial that nodes of $\mathrm{eval}(G)$ can be represented in polynomial space (with respect to the size of G) and hence can be guessed in polynomial time. This does not hold for non-linear SL cf tree grammars which can generate trees of doubly exponential size. Finally, one can use the fact that PSPACE is precisely the class of all problems that can be solved on an alternating Turing machine in polynomial time, cf. [18].

5 Open Problems and Conclusions

An interesting class of SL cf tree grammars that is missing in our present complexity analysis of tree automata is the class of *linear* SL cf tree grammar (with an unbounded number of parameters in contrast to Theorem 1). The results in this paper leave a gap from P to PSPACE for the uniform membership problem for TA and linear SL cf tree grammars (with an unbounded number of parameters). Our algorithm BPLEX from [9] outputs linear SL cf tree grammars. Note that BPLEX, even when bounding the number of parameters by a small constant (like 2 or 3), clearly outperforms compression by dags. The results presented here show that with respect to tree automata membership problems and XPath evaluation, exactly the same complexity bounds hold for linear SL cf tree grammars with a bounded number of parameter as for dags [5, 6]. This motivates us to believe that linear SL cf tree grammars are better suited than dags as memory efficient representations of XML documents. Precise trade-offs between the representations have to be determined in practice; we are currently implementing our ideas as part of BPLEX. For the XPath evaluation problem, the complexity for non-linear SL cf tree grammars remains open. We conjecture that the PSPACE upper bound from Theorem 9 cannot be generalized to the non-linear case.

References

1. Lohrey, M.: Word problems on compressed word. In Diaz, J., Karhumäki, J., Lepistö, A., Sannella, D., eds.: Proc. ICALP 2004, Turku (Finland). Number 3142 in Lecture Notes in Computer Science, Springer (2004) 906–918
2. Rytter, W.: Grammar compression, LZ-encodings, and string algorithms with implicit input. In Diaz, J., Karhumäki, J., Lepistö, A., Sannella, D., eds.: Proc. ICALP 2004, Turku (Finland). Number 3142 in Lecture Notes in Computer Science, Springer (2004) 15–27
3. Plump, D.: Term graph rewriting. In Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation. Volume 2. World Scientific (1999) 3–61
4. Bryant, R.E.: Symbolic boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys **24** (1992) 293–318
5. Buneman, P., Grohe, M., Koch, C.: Path queries on compressed XML. In Freytag, J.C., et al., eds.: Proc. VLDB 2003, Morgan Kaufmann (2003) 141–152
6. Frick, M., Grohe, M., Koch, C.: Query evaluation on compressed trees (extended abstract). In: Proc. LICS'2003, IEEE Computer Society Press (2003) 188–197
7. Maneth, S., Busatto, G.: Tree transducers and tree compressions. In Walukiewicz, I., ed.: Proc. FoSSaCS 2004, Barcelona (Spain). Number 2987 in Lecture Notes in Computer Science, Springer (2004) 363–377
8. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata` (2002)
9. Busatto, G., Lohrey, M., Maneth, S.: Efficient memory representation of XML documents. In: Proc. DBPL 2005, Trondheim (Norway), Springer (2005) to appear.
10. Gécseg, F., Steinby, M.: Tree automata. Akadémiai Kiadó (1984)
11. Murata, M., Lee, D., Mani, M.: Taxonomy of XML Schema Languages using Formal Language Theory. In: Proc. Extreme Markup Languages 2000, Montréal (Canada). (2000)
12. Neven, F.: Automata theory for XML researchers. SIGMOD Record **31** (2002) 39–46
13. Lohrey, M.: On the parallel complexity of tree automata. In Middeldorp, A., ed.: Proc. RTA 2001, Utrecht (The Netherlands). Number 2051 in Lecture Notes in Computer Science, Springer (2001) 201–215
14. Segoufin, L.: Typing and querying XML documents: some complexity bounds. In: Proc. PODS 2003, ACM Press (2003) 167–178
15. Downey, R.G., Fellows, M.R.: Parametrized Complexity. Springer (1999)
16. Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. In: Proc. VLDB 2002, Morgan Kaufmann (2002) 95–106
17. Gottlob, G., Koch, C., Pichler, R.: The complexity of XPath query evaluation. In: Proc. PODS 2003, ACM Press (2003) 179–190
18. Papadimitriou, C.H.: Computational Complexity. Addison Wesley (1994)
19. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)
20. Courcelle, B.: A representation of trees by languages I. Theoretical Computer Science **6** (1978) 255–279
21. Anantharaman, S., Narendran, P., Rusinowitch, M.: Closure properties and decision problems of dag automata. Information Processing Letters **94** (2005) 231–240
22. Markey, N., Schnoebelen, P.: A PTIME-complete matching problem for SLP-compressed words. Information Processing Letters **90** (2004) 3–6
23. Papadimitriou, C.H., Yannakakis, M.: On the complexity of database queries. Journal of Computer and System Sciences **58** (1999) 407–427