Tree Compression Using String Grammars[⋆]

Moses Ganardi, Danny Hucke, Markus Lohrey, and Eric Noeth

University of Siegen, Germany
{ganardi,hucke,lohrey,eric.noeth}@eti.uni-siegen.de

Abstract. We study the compressed representation of a ranked tree by a straight-line program (SLP) for its preorder traversal string, and compare it with the previously studied representation by straight-line context-free tree grammars (also known as tree straight-line programs or TSLPs). Although SLPs may be exponentially more succinct than TSLPs, we show that many simple tree queries can still be performed efficiently on SLPs, such as computing the height of a tree, tree navigation, or evaluation of Boolean expressions. Other problems like pattern matching and evaluation of tree automata become intractable.

1 Introduction

The idea of *grammar-based compression* is to represent a given string s by a small context-free grammar that generates only s; such a grammar is also called a *straight-line program* (SLP) for s. By repeated doubling, it is easy to produce a string of length 2^n by an SLP of size n (measured as the total length of all right-hand sides of the productions), i.e., exponential compression can be achieved in the best case. The goal of grammar-based compression is to construct from a given string s a small SLP for s. Whereas computing a smallest SLP for a given string is not possible in polynomial time unless $\mathsf{P} = \mathsf{NP}$ [9,28], there exist several linear time algorithms that produce grammars that are at worst $\mathcal{O}(\log(N/g))$ larger than the size of a smallest SLP, where N is the length of the input string s and g is the size of a smallest SLP for s [9,18,26].

Motivated by applications like XML processing, where large tree-structured data occur, grammar-based compression has been extended to trees, see [24] for a survey. Unless otherwise specified, a tree in this paper is always a rooted ordered tree over a ranked alphabet, i.e., every node is labelled with a symbol and the rank of this symbol is equal to the number of children of the node. This class of trees occurs in many different contexts like term rewriting, expression evaluation and tree automata. A tree over a ranked alphabet is uniquely represented by its preorder traversal. For instance, the preorder traversal of the tree $f(g(a), f(a, b))$ is the string $fgafab$. It is now a natural idea to apply a string compressor to this preorder traversal. In this paper we study the compression of ranked trees by SLPs for their preorder traversals. This idea is very similar to [6], where unranked unlabelled trees are compressed by SLPs for their balanced parenthesis representations.

[⋆] The third and fourth author are supported by the DFG-project LO 748/10-1.

In Section 3 we compare the size of SLPs for preorder traversals with other grammar-based compressed tree representations. SLPs for strings can also be generalized directly to trees, using context-free tree grammars that produce a single tree (so called tree straight-line programs, briefly TSLPs). TSLPs generalize dags (directed acyclic graphs), which are widely used as a compact tree representation. Whereas dags only allow to share repeated subtrees, TSLPs can also share repeated internal tree patterns. The algorithm from [13] produces for every tree over a fixed ranked alphabet a TSLP of size $\mathcal{O}(N/\log N)$, which is worst-case optimal. A grammar-based tree compressor using TSLPs with an approximation ratio of $\mathcal{O}(\log N)$ can be found in [19]. It was shown in [7] that from a given TSLP $\mathbb{A}$ of size m for a tree t one can efficiently construct an SLP of size $\mathcal{O}(m \cdot r)$ for the preorder traversal of t, where r is the maximal rank occurring in t (i.e. the maximal number of children of a node). Hence a smallest SLP for the traversal of t cannot be much larger than a smallest TSLP for t. Our first main result shows that SLPs can be exponentially more succinct than TSLPs: We construct a family of binary trees t_n $(n \geq 0)$ such that the size of a smallest SLP for the traversal of t_n is polynomial in n but the size of a smallest TSLP for t_n is $\Omega(2^{n/2})$. Moreover, we also construct a family of binary trees t_n $(n \geq 0)$ such that the size of a smallest SLP for the preorder traversal of t_n is polynomial in n but the size of a smallest SLP for the balanced parenthesis representation is $\Omega(2^{n/2})$. It remains open whether a family of trees with the opposite behavior exists.

We also study algorithmic problems for SLP-compressed trees. We extend some of the results from [6] on querying SLP-compressed balanced parenthesis representations to our context. Specifically, we show that after a linear time preprocessing we can navigate (i.e., move to the parent node and to the k^{th} child), compute lowest common ancestors and subtree sizes in time $\mathcal{O}(\log N)$, where N is the size of the tree represented by the SLP. For a couple of other problems (computation of the tree's height, the depth of a node and evaluation of Boolean expressions) we provide polynomial time algorithms for the case that the input tree is given by an SLP for the preorder traversal. On the other hand, there exist problems that are polynomial time solvable for TSLP-compressed trees but intractable for SLP-compressed trees: examples for such problems are pattern matching, evaluation of max-plus expressions, and membership for tree automata. Looking at tree automata is also interesting when compared with the situation for explicitly given (i.e., uncompressed) preorder traversals. For these, evaluating Boolean expressions (which is the membership problem for a particular tree automaton) is NC^1-complete by a famous result of Buss [8], and the NC^1 upper bound was generalized to every fixed tree automaton [21]. If we compress the preorder traversal by an SLP, the problem is still solvable in polynomial time for Boolean expressions (Thm. 13), but there is a fixed tree automaton with a PSPACE-complete evaluation problem (Thm. 16).

Missing proofs can be found in the long version [14].

Related work on tree compression. There are also tree compressors based on other grammar formalisms. In [1] so called elementary ordered tree grammars are used, and a polynomial time compressor with an approximation ratio of $\mathcal{O}(N^{5/6})$ is presented. Also the *top dags* from [5] can be seen as a variation of TSLPs for unranked trees. Recently, in [15] it was shown that for every tree of size N with σ many node labels, the top dag has size $\mathcal{O}(N \cdot \log\log_\sigma N / \log_\sigma N)$, which improved the bound from [5]. An extension of TSLPs to higher order tree grammars was proposed in [20].

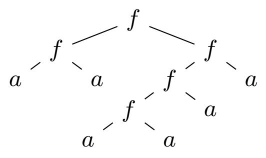
Another class of tree compressors use succinct data structures for trees. Here, the goal is to represent a tree in a number of bits that asymptotically matches the information theoretic lower bound, and at the same have efficient querying. For unlabelled (resp., node-labelled) unranked trees of size N there exist representations with $2N + o(N)$ bits (resp., $(2 + \log \sigma) \cdot N + o(N)$ bits, where σ is the number of node labels) that support navigation and some other tree queries in time $\mathcal{O}(1)$ [3,12,16,17,25].

2 Preliminaries

Let Σ be a finite alphabet. For a string $w = a_1 \cdots a_N \in \Sigma^*$ we define $|w| = N$, $w[i] = a_i$ and $w[i:j] = a_i \cdots a_j$ where $w[i:j] = \varepsilon$, if $i > j$. Let $w[:i] = w[1:i]$ and $w[i:] = w[i:|w|]$. With $\mathrm{rev}(w) = a_N \cdots a_1$ we denote w reversed. For $u, v \in \Sigma^*$, the *convolution* $u \otimes v \in (\Sigma \times \Sigma)^*$ is the string of length $\min\{|u|, |v|\}$ defined by $(u \otimes v)[i] = (u[i], v[i])$ for $1 \leq i \leq \min\{|u|, |v|\}$.

We assume familiarity with basic complexity classes like P, NP and PSPACE. The counting class $\#\mathsf{P}$ contains all functions $f : \Sigma^* \to \mathbb{N}$ for which there is a nondeterministic polynomial time machine M such that for all $x \in \Sigma^*$, $f(x)$ is the number of accepting computation paths of M on input x. The class PP contains all problems A for which there is a nondeterministic polynomial time machine M such that for all inputs x: $x \in A$ iff more than half of all computation paths of M on input x are accepting. When referring to linear time algorithms, we assume the standard RAM model of computation, where registers can hold hold numbers with $\mathcal{O}(\log n)$ bits for n the input size, and arithmetic operations on register values can be done in constant time.

A *ranked alphabet* $\mathcal{F}$ is a finite set of symbols, where every $f \in \mathcal{F}$ has a rank $\mathsf{rank}(f) \in \mathbb{N}$. By $\mathcal{F}_n$ we denote the symbols of $\mathcal{F}$ of rank n. We assume that $\mathcal{F}_0 \neq \emptyset$. Later we will also allow ranked alphabets where $\mathcal{F}_0$ is infinite. For the purpose of this paper, it is convenient to define trees as particular strings over the alphabet $\mathcal{F}$ (namely as preorder traversals). The set $\mathcal{T}(\mathcal{F})$ of all *trees* over $\mathcal{F}$ is the subset of $\mathcal{F}^*$ defined inductively as follows: If $f \in \mathcal{F}_n$ with $n \geq 0$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$, then also $ft_1 \cdots t_n \in \mathcal{T}(\mathcal{F})$ (we denote this tree also with $f(t_1, \ldots, t_n)$, which corresponds to the standard term notation). A string $s \in \mathcal{F}^*$ is a *fragment* if there exist a tree $t \in \mathcal{T}(\mathcal{F})$ and a non-empty string $x \in \mathcal{F}^+$ such that $sx = t$. Note that the empty string ε is a fragment. Intuitively, a fragment is a tree with gaps. For every non-empty fragment $s \in \mathcal{F}^+$ there is a unique $n \geq 1$ such that $\{x \in \mathcal{F}^* \mid sx \in \mathcal{T}(\mathcal{F})\} = (\mathcal{T}(\mathcal{F}))^n$; this n is denoted with $\mathsf{gaps}(s)$. We set $\mathsf{gaps}(\varepsilon) = 0$. Since $\mathcal{T}(\mathcal{F})$ is prefix-free we have:



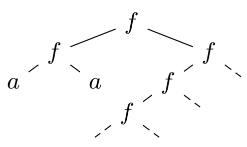


Fig. 1. The tree t from Example 2 and the tree fragment corresponding to $ffaafff$.

Lemma 1. *For every $w \in \mathcal{F}^*$ there exist unique $n \geq 0$, $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ and a unique fragment $s \in \mathcal{F}^*$ such that $w = t_1 \cdots t_n s$.*

Let $w \in \mathcal{F}^*$ and let $w = t_1 \cdots t_n s$ as in Lemma 1. We define $c(w) = (n, \mathsf{gaps}(s))$. The number n counts the number of full trees in w and $\mathsf{gaps}(s)$ is the number of trees that are missing in order to make the fragment s a tree.

We also consider trees in their graph-theoretic interpretation where the set of nodes of a tree t is the set of positions $\{1, \ldots, |t|\}$ of the string t. The root node is 1. If t factorizes as $uft_1 \cdots t_n v$ for $u, v \in \mathcal{F}^*$, $f \in \mathcal{F}_n$, and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$, then the n children of node $|u| + 1$ are $|u| + 2 + \sum_{i=1}^{k} |t_i|$ for $0 \leq k \leq n - 1$. We define the depth of a node in t (number of edges from the root to the node) and the height of t (maximal depth of a node) as usual. Note that the tree t as a string is simply the preorder traversal of the tree t seen in its standard graph-theoretic interpretation. Since for a ranked tree the number of children of a node is uniquely determined by the node label, a tree (in the above graph-theoretic interpretation) is uniquely determined by its preorder traversal and vice versa.

Example 2. Let $t = ffaafffaaaa = f(f(a,a), f(f(f(a,a),a),a))$ be the tree depicted in Fig. 1 with $f \in \mathcal{F}_2$ and $a \in \mathcal{F}_0$. Its height is 4. All prefixes (including the empty word, excluding the full word) of t are fragments. The fragment $s = ffaafff$ is also depicted in Fig. 1 in a graphical way. The dashed edges visualize the gaps. We have $\mathsf{gaps}(s) = 4$. For the factor $u = aafffa$ of t we have $c(u) = (2, 3)$. The children of node 5 (the third f-labelled node) are 6 and 11.

A *straight-line program*, briefly SLP, is a context-free grammar that produces a single string. Formally, it is a tuple $\mathbb{A} = (N, \Sigma, P, S)$, where N is a finite set of nonterminals, Σ is a finite set of terminals such that $\Sigma \cap N = \emptyset$, $S \in N$ is the start nonterminal, and P is a finite set of productions (or rules) of the form $A \to w$ for $A \in N$, $w \in (N \cup \Sigma)^*$ such that: (i) For every $A \in N$, there exists exactly one production of the form $A \to w$, and (ii) the binary relation $\{(A, B) \in N \times N \mid (A \to w) \in P,\ B \text{ occurs in } w\}$ is acyclic. Every nonterminal $A \in N$ produces a unique string $\mathsf{val}_{\mathbb{A}}(A) \in \Sigma^*$. The string defined by $\mathbb{A}$ is $\mathsf{val}(\mathbb{A}) = \mathsf{val}_{\mathbb{A}}(S)$. We omit the subscript $\mathbb{A}$ when it is clear from the context. The *size* of the SLP $\mathbb{A}$ is $|\mathbb{A}| = \sum_{(A \to w) \in P} |w|$. An SLP for a nonempty word can be transformed in linear time into *Chomsky normal form*, i.e., for each production $A \to w$, either $w \in \Sigma$ or $w = BC$ where $B, C \in N$. The following lemma summarizes known results about SLPs which we will use throughout the paper, see e.g. [23].

Lemma 3. *Let $\mathbb{A}$ be an SLP. There are algorithms running in time $\mathcal{O}(|\mathbb{A}|)$ for the following problems (the numbers i and j are given in binary encoding):*

1. *Compute the set of symbols occurring in* $\mathsf{val}(\mathbb{A})$.
2. *Let Σ be the terminal set of $\mathbb{A}$ and let $\Gamma \subseteq \Sigma$. Compute the number of occurrences of symbols from Γ in* $\mathsf{val}(\mathbb{A})$.
3. *Let Σ be the terminal set of $\mathbb{A}$ and let $\Gamma \subseteq \Sigma$. Given a number i, compute the position of the i^{th} occurrence of a symbol from Γ in* $\mathsf{val}(\mathbb{A})$ *(if it exists).*
4. *Given $1 \leq i, j \leq |\mathsf{val}(\mathbb{A})|$, compute an SLP of size $\mathcal{O}(|\mathbb{A}|)$ for* $\mathsf{val}(\mathbb{A})[i : j]$.

We want to compress trees (viewed as particular strings) by SLPs. This leads to the question whether a given SLP produces a tree, which is also known as the compressed membership problem for the language $\mathcal{T}(\mathcal{F}) \subseteq \mathcal{F}^*$. By computing bottom-up for each nonterminal A the pair $c(\mathsf{val}(A))$, we can show:

Theorem 4. *Given an SLP $\mathbb{A}$, one can check in time $\mathcal{O}(|\mathbb{A}|)$ whether* $\mathsf{val}(\mathbb{A}) \in \mathcal{T}(\mathcal{F})$.

Note that $\mathcal{T}(\mathcal{F})$ is context-free. In general the compressed membership problem for context-free languages belongs to PSPACE and there is a deterministic context-free language with a PSPACE-complete compressed membership problem [22].

Tree straight-line programs (briefly TSLPs) generalize SLPs to trees [13,19]. In addition to terminals and nonterminals, the productions of a TSLP also contain so called parameters $x_1, x_2, x_3, \ldots$, which are treated as symbols of rank zero (i.e., they only label leaves). Formally, a TSLP is a tuple $\mathbb{A} = (\mathcal{V}, \mathcal{F}, P, S)$, where $\mathcal{V}$ (resp., $\mathcal{F}$) is a ranked alphabet of nonterminals (resp., terminals), $S \in \mathcal{V}_0$ is the start nonterminal and P is a finite set of productions of the form $A(x_1, \ldots, x_n) \to t$ (which is also briefly written as $A \to t$), where $n \geq 0$, $A \in \mathcal{V}_n$ and $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{V} \cup \{x_1, \ldots, x_n\})$ is a tree in which every parameter x_i ($1 \leq i \leq n$) occurs at most once, such that: (i) For every $A \in \mathcal{V}_n$ there exists exactly one production of the form $A(x_1, \ldots, x_n) \to t$, and (ii) the binary relation $\{(A, B) \in \mathcal{V} \times \mathcal{V} \mid (A \to t) \in P, B \text{ is a label in } t\}$ is acyclic. These conditions ensure that exactly one tree $\mathsf{val}_{\mathbb{A}}(A) \in \mathcal{T}(\mathcal{F} \cup \{x_1, \ldots, x_n\})$ is derived from every nonterminal $A \in \mathcal{V}_n$ by using the rules as rewriting rules in the usual sense. As for SLPs, we omit the subscript $\mathbb{A}$ when the context is clear. The tree defined by $\mathbb{A}$ is $\mathsf{val}(\mathbb{A}) = \mathsf{val}_{\mathbb{A}}(S)$. The *size* $|\mathbb{A}|$ of a TSLP is the total number of non-parameter nodes in all right-hand sides of productions; see [13] for a justification of this. TSLPs in which every nonterminal has rank 0 correspond to dags (the nodes of the dag are the nonterminals of the TSLP).

3 Relative succinctness of SLP-compressed trees

In [7] it is shown that a TSLP $\mathbb{A}$ for a tree t can be transformed into an SLP of size $\mathcal{O}(|\mathbb{A}| \cdot r)$ for (the traversal of) t, where r is the maximal rank of a label in t. In this section we discuss the other direction, i.e., transforming an SLP into a

TSLP. For tree families of unbounded maximal rank, SLPs can trivially achieve exponentially better compression: The size of the smallest TSLP for $t_n = f_n a^n$ (with $f_n \in \mathcal{F}_n$) is $n+1$, whereas the size of the smallest SLP for t_n is in $\mathcal{O}(\log n)$. Note that this does not contradict the $\mathcal{O}(\frac{n}{\log n})$ bound from [13] since the trees t_n have unbounded rank. It is less obvious that such an exponential gap can also occur with trees of bounded rank. To show this, we use the following result:

Theorem 5 ([4, Thm. 2]). *For every $n > 0$, there exist words $u_n, v_n \in \{0,1\}^*$ with $|u_n| = |v_n|$ such that u_n and v_n have SLPs of size $n^{\mathcal{O}(1)}$, but the smallest SLP for the convolution $u_n \otimes v_n$ has size $\Omega(2^{n/2})$.*

For two words $u = i_1 \cdots i_n \in \{0,1\}^*$ and $v = j_1 \cdots j_n \in \{0,1\}^*$ we define the *comb tree* $t(u,v) = f_{i_1}(f_{i_2}(\dots f_{i_n}(\$, j_n)\dots j_2), j_1)$ over the ranked alphabet $\{f_0, f_1, 0, 1, \$\}$ where f_0, f_1 have rank 2 and $0, 1, \$$ have rank 0.

Theorem 6. *For every $n > 0$ there exists a tree t_n such that the size of a smallest SLP for t_n is polynomial in n, but the size of a smallest TSLP for t_n is in $\Omega(2^{n/2})$.*

Proof sketch. Let $t_n = t(u_n, v_n)$ be the comb tree, where u_n, v_n are from Thm. 5. These words have SLPs of size $n^{\mathcal{O}(1)}$, which yield an SLP of size $n^{\mathcal{O}(1)}$ for t_n. On the other hand, one can transform a TSLP for t_n of size m into an SLP of size $\mathcal{O}(m)$ for $u_n \otimes v_n$, which implies the result. □

Note that the height of the tree t_n in Thm. 6 is linear in the size of t_n. By the following result, large height and rank are always responsible for the exponential succinctness gap between SLPs and TSLPs.

Theorem 7. *Let $t \in \mathcal{T}(\mathcal{F})$ be a tree of height h and maximal rank r, and let $\mathbb{A}$ be an SLP for t. Then there exists a TSLP $\mathbb{B}$ with $\mathsf{val}(\mathbb{B}) = t$ such that $|\mathbb{B}| \in \mathcal{O}(|\mathbb{A}| \cdot h \cdot r)$, which can be constructed in time $\mathcal{O}(|\mathbb{A}| \cdot h \cdot r)$.*

Proof sketch. Without loss of generality we assume that $\mathbb{A}$ is in Chomsky normal form. Consider a nonterminal A of $\mathbb{A}$ with $c(A) = (a_1, a_2)$. This means that $\mathsf{val}(A) = t_1 \cdots t_{a_1} s$, where the t_i is a full tree and s is a fragment with a_2 many gaps. For the TSLP $\mathbb{B}$, we introduce (i) a_1 nonterminals $A_1, \dots, A_{a_1}$ of rank 0, which produce the trees $t_1, \dots, t_{a_1}$, and (ii), if $a_2 > 0$, one nonterminal A' of rank a_2 for the fragment s. For every rule of the form $A \to f$ with $f \in \mathcal{F}_n$ we add to $\mathbb{B}$ the TSLP-rule $A_1 \to f$ if $n = 0$ or $A'(x_1, \dots, x_n) \to f(x_1, \dots, x_n)$ if $n \geq 1$. For a rule of the form $A \to BC$ with $c(B) = (b_1, b_2)$ and $c(C) = (c_1, c_2)$, one has to distinguish the cases $b_2 = 0$, $0 < b_2 \leq c_1$, and $b_2 > c_1$. In each of these cases it is straightforward to define the rules in such a way that A_i derives t_i and, in case $a_2 > 0$), A' produces the fragment s. Finally, it is easy to achieve the size bound $\mathcal{O}(|\mathbb{A}| \cdot h \cdot r)$ in the construction for $\mathbb{B}$. □

Balanced parenthesis sequences are widely used as a succinct representation of ordered unranked unlabelled trees [25]. One defines the balanced parenthesis sequence $\mathsf{bp}(t)$ of such a tree t inductively as follows. If t consists of a single node, then $\mathsf{bp}(t) = ()$. If the root of t has n children in which the subtrees $t_1, \ldots, t_n$ are rooted (from left to right), then $\mathsf{bp}(t) = (\mathsf{bp}(t_1) \cdots \mathsf{bp}(t_n))$. Using a construction similar to the proof of Thm. 6 we can show:

Theorem 8. *For every $n > 0$ there exists a binary tree $t_n \in \mathcal{T}(\{a, f\})$ (where f has rank 2 and a has rank 0) such that the size of a smallest SLP for t_n is polynomial in n, but the size of a smallest SLP for $\mathsf{bp}(t_n)$ is in $\Omega(2^{n/2})$.*

It remains open whether there is also a family of trees where the opposite situation arises, i.e., where a smallest SLP for the balanced parenthesis sequence is exponentially smaller than a smallest SLP for the preorder traversal.

4 Algorithmic problems on SLP-compressed trees

For trees given by TSLPs or other compressed representations, various algorithmic questions have been studied in the literature [5,15,24,27]. Here, we study the complexity of several basic algorithmic problems on trees that are represented by SLPs. In this context the main difficulty for SLPs in contrast to TSLPs is that the tree structure is only given implicitly by the ranked alphabet.

4.1 Tree navigation and pattern matching

In [6] it is shown that from an SLP of size n that produces the balanced parenthesis representation of an unranked tree t of size N, one can compute in time $\mathcal{O}(n)$ a data structure of size $\mathcal{O}(n)$ that supports navigation as well as other important computations (e.g. lowest common ancestors) in time $\mathcal{O}(\log N)$. Here, the word RAM model is used, where memory cells can store numbers with $\log N$ bits and arithmetic operations on $\log N$-bit numbers can be carried out in constant time. An analogous result was shown in [5] for top dags. Here, we show the same result for SLPs that produce (preorder traversals of) ranked trees. Recall that we identify the nodes of a tree t with the positions $1, \ldots, |t|$ in the string t. The proof of the following result combines results from [3,6,17] and uses a correspondence between preorder traversals of ranked trees and the DFUDS (depth-first unary-degree sequence) representation of unranked trees from [3].

Theorem 9. *Given an SLP of size n for a tree t of size N, one can produce in time $\mathcal{O}(n)$ a data structure of size $\mathcal{O}(n)$ that allows to do the following computations in time $\mathcal{O}(\log N) \leq \mathcal{O}(n)$, where $i, j, k \in \mathbb{N}$ with $1 \leq i, j \leq N$ are given in binary notation:*

(a) Compute the parent node of node $i > 1$ in t.
(b) Compute the k^{th} child of node i in t, if it exists.
(c) Compute the number k such that $i > 1$ is the k^{th} child of its parent node.
(d) Compute the size of the subtree rooted at node i.
(e) Compute the lowest common ancestor of node i and j in t.

The data structure of [6] allows to compute the height and the depth of a given tree node in time $\mathcal{O}(\log N)$ as well. It is not clear to us whether this result also can be extended to our setting. On the other hand, in Section 4.2, we show that the height and the depth of a given node of an SLP-compressed tree can be computed in polynomial time.

In contrast to navigation, simple pattern matching problems are intractable for SLP-compressed trees. The *pattern matching problem for SLP-compressed trees* is defined as follows: Given a tree $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{X})$ (the *pattern*), where every variable $x \in \mathcal{X}$ (a symbol of rank zero) occurs at most once, and an SLP $\mathbb{A}$ producing a tree $t \in \mathcal{T}(\mathcal{F})$, is there a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F})$ such that $\sigma(s)$ is a subtree of t? Here, $\sigma(s) \in \mathcal{T}(\mathcal{F})$ denotes the tree obtained from s by substituting each variable $x \in \mathcal{X}$ by $\sigma(x)$. Note that the pattern is given uncompressed. If the tree t is given by a TSLP, the corresponding problem can be solved in polynomial time [27].[1] For SLP-compressed trees we have:

Theorem 10. *The pattern matching problem for SLP-compressed trees is* NP*-complete. Moreover,* NP*-hardness holds for a fixed pattern of the form* $f(x, a)$*.*

NP-hardness is shown by a reduction from the question whether $(1, 1)$ appears in the convolution of two SLP-compressed strings over $\{0, 1\}$ [23, Thm. 3.13].

4.2 Tree evaluation problems

The algorithmic difficulty of SLP-compressed trees already becomes clear when computing the height. For TSLPs it is easy to see that the height of the produced tree can be computed in linear time: Compute bottom-up for each nonterminal the height of the produced tree and the depths of the parameter nodes. However, this direct approach fails for SLPs since each nonterminal encodes a possibly exponential number of trees. The crucial observation to solve this problem is that one can store and compute the required information for each nonterminal in a compressed form.

In the following we present a general framework to define and solve evaluation problems on SLP-compressed trees. We assign to each alphabet symbol of rank n an n-ary operator which defines the value of a tree by evaluating it bottom-up. This approach includes natural tree problems like computing the height of a tree, evaluating a Boolean expression or determining whether a fixed tree automaton accepts a given tree. We only consider operators on $\mathbb{Z}$ but other domains with an appropriate encoding of the elements are also possible. To be able to consider arbitrary arithmetic expressions properly, it is necessary to allow the set $\mathcal{F}_0 \subseteq \mathcal{F}$ of constants to be an infinite subset of $\mathbb{Z}$. If such a constant $a \in \mathcal{F}_0$ appears in an SLP for a tree, then its contribution to the SLP size is the number of bits of the binary representation of a.

[1] In fact, there is a polynomial time algorithm that checks whether a TSLP-compressed pattern tree s occurs in a TSLP-compressed tree t [27]. But for this, it is important that every variable x occurs at most once in the pattern s. For the case that variables are allowed to occur repeatedly in the pattern, the precise complexity is open.

Let $\mathcal{D} \subseteq \mathbb{Z}$ be a possibly infinite set of integers and let $\mathcal{F}$ be a ranked alphabet with $\mathcal{F}_0 = \mathcal{D}$. An *interpretation $\mathcal{I}$ of $\mathcal{F}$ over $\mathcal{D}$* assigns to each symbol $f \in \mathcal{F}_n$ an n-ary function $f^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{D}$ with the restriction that $a^{\mathcal{I}} = a$ for all $a \in \mathcal{D}$. We lift the definition of $\mathcal{I}$ to $\mathcal{T}(\mathcal{F})$ inductively by $(f\, t_1 \cdots t_n)^{\mathcal{I}} = f^{\mathcal{I}}(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}})$, where $f \in \mathcal{F}_n$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$. The problem *$\mathcal{I}$-evaluation for SLP-compressed trees* is: Given an SLP $\mathbb{A}$ over $\mathcal{F}$ with $\mathsf{val}(\mathbb{A}) \in \mathcal{T}(\mathcal{F})$, compute $\mathsf{val}(\mathbb{A})^{\mathcal{I}}$.

In a first step, we reduce $\mathcal{I}$-evaluation for SLP-compressed trees to the corresponding problem for SLP-compressed caterpillar trees. A tree $t \in \mathcal{T}(\mathcal{F})$ is called a *caterpillar tree* if every node has at most one child which is not a leaf. Let $s \in \mathcal{F}^*$ be an arbitrary string. Then $s^{\mathcal{I}} \in \mathcal{F}^*$ denotes the unique string obtained from s by replacing every maximal substring $t \in \mathcal{T}(\mathcal{F})$ of s by its value $t^{\mathcal{I}}$. By Lemma 1 we can factorize s uniquely as $s = t_1 \cdots t_n u$ where $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ and u is a fragment. Hence $s^{\mathcal{I}} = m_1 \cdots m_n u^{\mathcal{I}}$ with $m_1, \ldots, m_n \in \mathcal{D}$. Since u is a fragment, the string $u^{\mathcal{I}}$ is the fragment of a caterpillar tree (briefly, *caterpillar fragment*). For instance, with the standard interpretation of $+$ and $\times$ on integers, we have $(0, 2, +, 2, +, +, \times, 2, +, 2, 1, +, \times)^{\mathcal{I}} = 0, 2, +, 2, +, +, 6, +, \times$ (commas are added for better readability).

Our reduction to caterpillar trees only works for interpretations $\mathcal{I}$ that are *polynomially bounded* in the following sense: There exist constants $\alpha, \beta \geq 0$ such that for every tree $t \in \mathcal{T}(\mathcal{F})$, $\mathsf{abs}(t^{\mathcal{I}}) \leq \left(\beta \cdot |t| + \sum_{i \in L} \mathsf{abs}(t[i])\right)^{\alpha}$, where $\mathsf{abs}(z)$ is the absolute value of $z \in \mathbb{Z}$ (we write $\mathsf{abs}(z)$ instead of $|z|$ in order to not get confused with the size $|t|$ of a tree) and $L \subseteq \{1, \ldots, |t|\}$ is the set of leaves of t. The purpose of this definition is to ensure that for every SLP $\mathbb{A}$ for a tree t, the length of the binary encoding of $t^{\mathcal{I}}$ is polynomially bounded in $|\mathbb{A}|$ and the binary lengths of the integer constants that appear in $\mathbb{A}$.

Theorem 11. *Let $\mathcal{I}$ be a polynomially bounded interpretation. Then the $\mathcal{I}$-evaluation for SLP-compressed trees is polynomial time Turing-reducible to the $\mathcal{I}$-evaluation for SLP-compressed caterpillar trees.*

Proof. In the proof we use an extension of SLPs by the cut-operator, called *composition systems*. A *composition system* $\mathbb{A} = (N, \Sigma, P, S)$ is an SLP where P may also contain rules of the form $A \to B[i : j]$ where $A, B \in N$ and $i, j \geq 0$. Here we let $\mathsf{val}(A) = \mathsf{val}(B)[i : j]$. It is known (see e.g. [23]) that a given composition system can be transformed in polynomial time into an SLP with the same value. We may also use more complex rules like for instance $A \to B[i : j]C[k : l]$. Such rules can be easily reduced to the above format.

Let $\mathbb{A} = (N, \mathcal{F}, P, S)$ be the input SLP in Chomsky normal form. We compute a composition system, which contains for each nonterminal $A \in N$ two nonterminals A_1 and A_2 such that the following holds: Assume that $\mathsf{val}(A) = t_1 \cdots t_n\, s$, where $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ and s is a fragment (hence $c(\mathsf{val}(A)) = (n, \mathsf{gaps}(s))$). Then we will have $\mathsf{val}(A_1) = t_1^{\mathcal{I}} \cdots t_n^{\mathcal{I}} \in \mathcal{D}^*$ and $\mathsf{val}(A_2) = s^{\mathcal{I}}$. In particular, $\mathsf{val}(A_1)\mathsf{val}(A_2) = \mathsf{val}(A)^{\mathcal{I}}$ and $\mathsf{val}(\mathbb{A})^{\mathcal{I}}$ is given by a single number in $\mathsf{val}(S_1)$. The fact that $\mathcal{I}$ is polynomially bounded ensures that all numbers $t_i^{\mathcal{I}}$ as well as all numbers that appear in the caterpillar tree $s^{\mathcal{I}}$ have polynomially many bits in the input length $|\mathbb{A}|$.

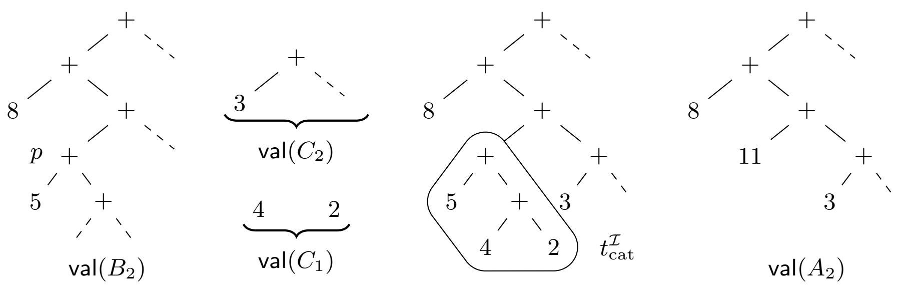


Fig. 2. An example for the case $b_2 > c_1$ in the proof of Thm. 11 ($+$ is interpreted as addition). Inserting the values from $\mathsf{val}(C_1) = 4\,2$ into the caterpillar fragment $\mathsf{val}(B_2) = + + 8 + + 5 +$ produces a caterpillar subtree t_{cat}, which evaluates to 11. Then, the fragment $\mathsf{val}(C_2) = +3$ is appended, which yields $\mathsf{val}(A_2) = + + 8 + 11 + 3$.

The computation is straightforward for rules $A \to f$ with $A \in N$ and $f \in \mathcal{F}$: If $\mathsf{rank}(f) = 0$, then $\mathsf{val}(A_1) = f$ and $\mathsf{val}(A_2) = \varepsilon$. If $\mathsf{rank}(f) > 0$, then $\mathsf{val}(A_1) = \varepsilon$ and $\mathsf{val}(A_2) = f$. For a nonterminal $A \in N$ with the rule $A \to BC$ we make a case distinction depending on $c(\mathsf{val}(B)) = (b_1, b_2)$ and $c(\mathsf{val}(C)) = (c_1, c_2)$.

Case $b_2 \le c_1$: Then concatenating $\mathsf{val}(B)$ and $\mathsf{val}(C)$ yields a new tree t_{new} (or ε if $b_2 = 0$) in $\mathsf{val}(A)$. Notice that $t^{\mathcal{I}}_{\mathrm{new}}$ is the value of the tree $\mathsf{val}(B_2)\,\mathsf{val}(C_1)[:\,b_2]$. Hence we can compute $t^{\mathcal{I}}_{\mathrm{new}}$ in polynomial time by computing an SLP that produces $\mathsf{val}(B_2)\,\mathsf{val}(C_1)[:\,b_2]$ and querying the oracle for caterpillar trees. We add the rules $A_1 \to B_1\, t^{\mathcal{I}}_{\mathrm{new}}\, C_1[b_2+1 : c_1]$, $A_2 \to C_2$ to the composition system.

Case $b_2 > c_1$: Then all trees and the fragment produced by C are inserted into the gaps of the fragment encoded by B. If $c_1 = 0$ (i.e., $\mathsf{val}(C_1) = \varepsilon$), then we add the productions $A_1 \to B_1$ and $A_2 \to B_2 C_2$. Now assume that $c_1 > 0$. Consider the fragment $s = \mathsf{val}(B_2)\,\mathsf{val}(C_1)\,\mathsf{val}(C_2)$. Intuitively, this fragment s is obtained by taking the caterpillar fragment $\mathsf{val}(B_2)$, where the first c_1 many gaps are replaced by the constants from the sequence $\mathsf{val}(C_1)$ and the $(c_1+1)^{\mathrm{st}}$ gap is replaced by the caterpillar fragment $\mathsf{val}(C_2)$, see Figure 2 for an example. If s is not already a caterpillar fragment, then we have to replace the (unique) largest factor of s which belongs to $\mathcal{T}(\mathcal{F})$ by its value under $\mathcal{I}$ to get $s^{\mathcal{I}}$. To do so we proceed as follows: Consider the tree $t' = \mathsf{val}(B_2)\,\mathsf{val}(C_1)\,\diamond^{b_2-c_1}$, where $\diamond$ is an arbitrary symbol of rank 0, and let $r = |\mathsf{val}(B_2)|+c_1+1$ (the position of the first $\diamond$ in t'). Let q be the parent node of r, which can be computed in polynomial time by Thm. 9. Using Lemma 3 we compute the position p (which is marked in the left tree in Figure 2) of the first occurrence of a symbol in $t'[q+1:]$ with rank > 0. If no such symbol exists, then s is already a caterpillar fragment and we add the rules $A_1 \to B_1$ and $A_2 \to B_2 C_1 C_2$ to the composition system. Otherwise p is the first symbol of the largest factor from $\mathcal{T}(\mathcal{F})$ described above. Using Thm. 9(d), we can compute in polynomial time the last position p' of the subtree of t' that is rooted in p. Note that the position p must belong to $\mathsf{val}(B_2)$ and that p' must belong to $\mathsf{val}(C_1)$ (since $c_1 > 0$). The string $t_{\mathrm{cat}} = (\mathsf{val}(B_2)\,\mathsf{val}(C_1))[p : p']$ is a caterpillar tree for which we can compute an SLP in polynomial time by the above remark on composition systems. Hence, using the oracle we can compute the value $t_{\mathrm{cat}}^{\mathcal{I}}$. We then add the rules $A_1 \to B_1$, $A' \to B_2 C_1$, and $A_2 \to A'[:p-1]\, t_{\mathrm{cat}}^{\mathcal{I}}\, A'[p'+1:]\, C_2$ to the composition system. This completes the proof. □

Polynomial time solvable evaluation problems. Next, we present several applications of Thm. 11. We start with the height of a tree.

Theorem 12. *The height of a tree $t \in \mathcal{T}(\mathcal{F})$ given by an SLP and the depth of a given node in t can be computed in polynomial time.*

Proof. We can assume that t is not a single constant. We replace every symbol in $\mathcal{F}_0$ by the integer 0. Then the height of t is given by its value under the interpretation $\mathcal{I}$ with $f^{\mathcal{I}}(a_1, \ldots, a_n) = 1 + \max\{a_1, \ldots, a_n\}$ for symbols $f \in \mathcal{F}_n$ with $n > 0$. Clearly $\mathcal{I}$ is polynomially bounded. By Thm. 11 it is enough to show how to evaluate a caterpillar tree t given by an SLP $\mathbb{A}$ in polynomial time under the interpretation $\mathcal{I}$. But note that arbitrary natural numbers may occur at leaf positions in this caterpillar tree.

Let $\mathcal{D}_t = \{d \in \mathbb{N} \mid d \text{ labels a leaf of } t\}$. The size of this set is bounded by $|\mathbb{A}|$. For $d \in \mathcal{D}_t$ let v_d be the deepest node such that d is the label of a child of node v_d (in particular, v_d is not a leaf). Let us first argue that v_d can be computed in polynomial time: Let k be the maximal position in t where a symbol of rank larger than zero occurs. The number k is computable in polynomial time by Lemma 3 (point 2 and 3). Again using Lemma 3 we compute the position of d's last (resp., first) occurrence in $t[:k]$ (resp., $t[k+1:]$). Then using Thm. 9 we compute the parent nodes of those two nodes. The larger (i.e., deeper one) is v_d.

Assume that $\mathcal{D}_t = \{d_1, \ldots, d_m\}$, where w.l.o.g. $v_{d_1} < v_{d_2} < \cdots < v_{d_m}$ (if $v_{d_i} = v_{d_j}$ for $d_i < d_j$, then we simply ignore d_i in the following consideration). Note that v_{d_m} is the maximal position in t where a symbol of rank at least one occurs (called k above) and that all children of v_{d_m} are labelled with d_m. Let t_i be the subtree rooted at v_{d_i}. Then $t_m^{\mathcal{I}} = d_m + 1$. We claim that from the value $t_{i+1}^{\mathcal{I}}$ we can compute in polynomial time the value $t_i^{\mathcal{I}}$. The crucial point is that all constants that appear in the interval $[v_{d_i}+1, v_{d_{i+1}}-1]$ except for d_i have a deeper occurrence in the tree and therefore can be ignored for the evaluation under $\mathcal{I}$. More precisely, if a is the number of occurrences of symbols of rank at least one in the interval $[v_{d_i}+1, v_{d_{i+1}}-1]$ (which can be computed in polynomial time by Lemma 3), then $t_i^{\mathcal{I}} = 1 + \max\{t_{i+1}^{\mathcal{I}} + a, d_i\}$. Finally, using the same argument, we can compute $t^{\mathcal{I}}$ from $t_1^{\mathcal{I}}$.

For the second part of the theorem, the computation of the depth of a given node can be easily reduced to a height computation. □

In the full version [14], we show with similar arguments that also the *Horton-Strahler number* [11] of an SLP-compressed tree can be computed in polynomial time. It can be defined as the value $t^{\mathcal{I}}$ under the interpretation $\mathcal{I}$ over $\mathbb{N}$ which interprets constant symbols $a \in \mathcal{F}_0$ by $a^{\mathcal{I}} = 0$ and each symbol $f \in \mathcal{F}_n$ with $n > 0$ as follows: Let $a_1, \ldots, a_n \in \mathbb{N}$ and $a = \max\{a_1, \ldots, a_n\}$. We set $f^{\mathcal{I}}(a_1, \ldots, a_n) = a$ if exactly one of $a_1, \ldots, a_n$ is equal to a, and otherwise $f^{\mathcal{I}}(a_1, \ldots, a_n) = a + 1$.

If the interpretation $\mathcal{I}$ is clear from the context, we also speak of the problem of *evaluating SLP-compressed $\mathcal{F}$-trees*. In the following theorem the interpretation is given by the Boolean operations $\wedge$ and $\vee$ over $\{0, 1\}$.

Theorem 13. *SLP-compressed $\{\wedge, \vee, 0, 1\}$-trees can be evaluated in polynomial time.*

Difficult arithmetical evaluation problems. Assume that $\mathcal{I}$ is the interpretation that assigns to the binary symbols max, $+$, and $\times$ their standard meanings over $\mathbb{Z}$. We consider the problem of evaluating SLP-compressed expressions over $\{\max, +\}$ or $\{+, \times\}$. For circuits, these problems are well-studied. Circuits over max and $+$ can be evaluated bottom-up in polynomial time, since all values that arise in the circuit only need polynomially many bits. Circuits are dags, and the latter correspond to TSLPs where all nonterminals have rank 0. Moreover, it was shown in [13] that a TSLP that evaluates to an expression over a semiring can be transformed in polynomial time into an equivalent circuit over the same semiring. Hence, TSLPs over max and $+$ can be evaluated in polynomial time. In contrast, for SLP-compressed expressions we can show the following result. The counting hierarchy CH is a hierarchy of complexity classes within PSPACE, and it is conjectured that $\mathsf{CH} \subsetneq \mathsf{PSPACE}$, see [2] for more details.

Theorem 14. *The evaluation of SLP-compressed $(\{\max, +\} \cup \mathbb{Z})$-trees belongs to CH and is $\#\mathsf{P}$-hard (even for SLP-compressed $(\{\max, +\} \cup \mathbb{N})$-trees).*

For expressions over $+$ and $\times$ the situation is more difficult. Clearly a circuit of size $\mathcal{O}(n)$ can produce the number 2^{2^n} which has 2^n bits. Hence, we cannot evaluate a circuit over $+$ and $\times$ in polynomial time. In [2] it was shown that the problem BitSLP of computing the k^{th} bit (k is given in binary) of the number to which a given arithmetic circuit evaluates to belongs to CH and is $\#\mathsf{P}$-hard. By [13] these results also hold for TSLPs. For the related problem PosSLP of deciding, whether a given arithmetic circuit computes a positive number, no non-trivial lower bound is known, see also [2]. For SLP-compressed expressions over $+$ and $\times$ we can show the following:

Theorem 15. *The problem of computing for a given binary encoded number k and an SLP $\mathbb{A}$ over $\{+, \times\} \cup \mathbb{Z}$ the k^{th} bit of $\mathsf{val}(\mathbb{A})^{\mathcal{I}}$ belongs to CH. Moreover, the problem of checking $\mathsf{val}(\mathbb{A})^{\mathcal{I}} \geq 0$ is PP-hard.*

Tree automata. A *(deterministic) tree automaton* $\mathcal{A} = (Q, \mathcal{F}, \Delta, F)$ consists of a finite set of *states* Q, a ranked alphabet $\mathcal{F}$, a set of *final states* $F \subseteq Q$ and a set Δ of *transition rules*, which contains for all $f \in \mathcal{F}_n$, $q_1, \ldots, q_n \in Q$ exactly one rule $f(q_1, \ldots, q_n) \to q$. A tree $t \in \mathcal{T}(\mathcal{F})$ is *accepted* by $\mathcal{A}$ if $t \xrightarrow{*}_{\Delta} q$ for some $q \in F$ where $\to_{\Delta}$ is the rewriting relation defined by Δ as usual. See [10] for more details on tree automata. One can also define nondeterministic tree automata, but the above deterministic model fits better into our framework: A tree automaton as defined above can be seen as a finite algebra (i.e., an interpretation $\mathcal{I}$, where the domain $\mathcal{D}$ is finite): The domain of the algebra is the set of states, and the operations of the algebra correspond to the transitions of the automaton. Then, the membership problem for the tree automaton corresponds to the evaluation problem in the finite algebra. The uniform membership problem for tree automata asks whether a given tree automaton accepts a given tree. In [21] it was shown that this problem belongs to $\mathsf{LogDCFL} \subseteq \mathsf{P}$ (for nondeterministic tree automata it becomes LogCFL-complete). For every fixed tree automaton, the membership problem belongs to NC^1 [21] if the tree is represented by its traversal string. If the input tree is given by a TSLP, the uniform membership problem becomes P-complete [24]. For SLP-compressed trees we have:

Theorem 16. *Uniform membership for tree automata is* PSPACE*-complete if the input tree is given by an SLP. Moreover,* PSPACE*-hardness holds for a fixed tree automaton.*

Proof sketch. For the upper bound one uses the fact that the uniform membership problem for explicitly given trees can be solved in $\mathsf{LogDCFL} \subseteq \mathsf{DSPACE}(\log^2(n))$. Given an SLP $\mathbb{A}$ for the tree $t = \mathsf{val}(\mathbb{A})$, one can run the $\mathsf{DSPACE}(\log^2(n))$-algorithm on the tree t without producing the whole tree t (which does not fit into polynomial space) before. This leads to a polynomial space algorithm.

For the lower bound we use a fixed regular language $L \subseteq (\{0,1\}^2)^*$ from [22] such that the following problem is PSPACE-complete: Given SLPs $\mathbb{A}$ and $\mathbb{B}$ over $\{0,1\}$ with $|\mathsf{val}(\mathbb{A})| = |\mathsf{val}(\mathbb{B})|$, is $\mathsf{val}(\mathbb{A}) \otimes \mathsf{val}(\mathbb{B}) \in L$? It is straightforward to transform a finite automaton for the fixed language L into a tree automaton $\mathcal{A}$ such that $\mathcal{A}$ accepts the comb tree $t(\mathrm{rev}(u), \mathrm{rev}(v))$ with $u = \mathsf{val}(\mathbb{A})$ and $v = \mathsf{val}(\mathbb{B})$ iff $u \otimes u \in L$. □

Thm. 16 implies that there exists a fixed finite algebra for which the evaluation problem for SLP-compressed trees is PSPACE-complete. This is somewhat surprising if we compare the situation with dags or TSLP-compressed trees. For these, membership for tree automata is still doable in polynomial time [24], whereas the evaluation problem of arithmetic expressions (in the sense of computing a certain bit of the output number) belongs to the counting hierarchy and is $\#\mathsf{P}$-hard. In contrast, for SLP-compressed trees, the evaluation problem for finite algebras (i.e., tree automata) is harder than the evaluation problem for arithmetic expressions (PSPACE versus the counting hierarchy).

References

1. T. Akutsu. A bisection algorithm for grammar-based compression of ordered trees. *Inf. Process. Lett.*, 110(18-19):815–820, 2010.
2. E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.
3. D. Benoit, E. D. Demaine, J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005.
4. A. Bertoni, C. Choffrut, and R. Radicioni. Literal shuffle of compressed words. In *Proc. IFIP TCS 2008*, volume 273 of *IFIP*, pages 87–100. Springer, 2008.
5. P. Bille, I. L. Gørtz, G. M. Landau, and O. Weimann. Tree compression with top trees. *Inform. Comput.*, 243: 166–177, 2015.

6. P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. R. Satti, and O. Weimann. Random access to grammar-compressed strings and trees. *SIAM J. Comput.*, 44(3)513–539:, 2015.
7. G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Inform. Syst.*, 33(4–5):456–474, 2008.
8. Samuel R. Buss. The Boolean formula value problem is in ALOGTIME. In *Proc. STOC 87*, pages 123–131. ACM Press, 1987.
9. M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005.
10. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree automata techniques and applications. `tata.gforge.inria.fr/`.
11. J. Esparza, M. Luttenberger, and M. Schlund. A brief history of strahler numbers. In *Proc. LATA 2014*, volume 8370 of *LNCS*, pages 1–13. Springer, 2014.
12. P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1), 2009.
13. M. Ganardi, D. Hucke, A. Jeż, M. Lohrey, and E. Noeth. Constructing small tree grammars and small circuits for formulas. arXiv.org, 2014. `arxiv.org/abs/1407.4286`.
14. M. Ganardi, D. Hucke, M. Lohrey, and E. Noeth. Tree compression using string grammars. arXiv.org, 2014. `arxiv.org/abs/1504.05535`.
15. L. Hübschle-Schneider and R. Raman. Tree compression with top trees revisited. In *Proc. SEA 2015*, volume 9125 of *LNCS*, pages 15–27. Springer, 2015.
16. G. Jacobson. Space-efficient static trees and graphs. In *Proc. FOCS 1989*, pages 549–554. IEEE Computer Society, 1989.
17. J. Jansson, K. Sadakane, and W-K. Sung. Ultra-succinct representation of ordered trees with applications. *J. Comput. Syst. Sci.*, 78(2):619–631, 2012.
18. A. Jeż. Approximation of grammar-based compression via recompression. In *Proc.CPM 2013*, volume 7922 of *LNCS*, pages 165–176. Springer, 2013.
19. A. Jeż and M. Lohrey. Approximation of smallest linear tree grammars. In *Proc. STACS 2014*, volume 25 of *LIPIcs*, pages 445–457. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.
20. N. Kobayashi, K. Matsuda, and A. Shinohara. Functional programs as compressed data. In *Proc. PEPM 2012*, pages 121–130. ACM Press, 2012.
21. M. Lohrey. On the parallel complexity of tree automata. In *Proc. RTA 2001*, volume 2051 of *LNCS*, pages 201–215. Springer, 2001.
22. M. Lohrey. Leaf languages and string compression. *Inform. Comput.*, 209(6):951–965, 2011.
23. M. Lohrey. *The Compressed Word Problem for Groups.* Springer, 2014.
24. M. Lohrey. Grammar-based tree compression. In *Proc. DLT 2015*, volume 9168 of *LNCS*, pages 46–57. Springer, 2015.
25. J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31(3):762–776, 2001.
26. W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1–3):211–222, 2003.
27. M. Schmidt-Schauß. Linear compressed pattern matching for polynomial rewriting. In *Proc. TERMGRAPH 2013*, volume 110 of *EPTCS*, pages 29–40, 2013.
28. J. A. Storer and T. G. Szymanski. The macro model for data compression. In *Proc. STOC 1978*, pages 30–39. ACM, 1978.