Noname manuscript No.
(will be inserted by the editor)

Tree Compression Using String Grammars

Moses Ganardi · Danny Hucke · Markus Lohrey · Eric Noeth

the date of receipt and acceptance should be inserted later

Abstract We study the compressed representation of a ranked tree by a (string) straight-line program (SLP) for its preorder traversal, and compare it with the well-studied representation by straight-line context free tree grammars (which are 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 on tree traversals turn out to be intractable, e.g. pattern matching and evaluation of tree automata. These problems can be still solved in polynomial time for TSLPs.

1 Introduction

Grammar-based compression has become an active field in string compression during the past 20 years. The idea 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 instance, the word $(ab)^{1024}$ can be represented by the SLP with the productions $A_0 \to ab$ and $A_i \to A_{i-1}A_{i-1}$ for $1 \leq i \leq 10$ (A_{10} is the start symbol). The size of this SLP (the size of an SLP is usually defined as the total length of all right-hand sides of the productions) is much smaller than the length of the string $(ab)^{1024}$. In general, an SLP of size n can produce a string of length $2^{\Omega(n)}$. Hence, an SLP can be seen indeed as a succinct representation of the generated string. The goal of grammar-based string compression is to construct from a given input string s a small SLP that produces s. Several algorithms for this have been proposed and

The third and fourth author are supported by the DFG-project LO 748/10-1 (QUANT-KOMP).

Universität Siegen, Germany
E-mail: {ganardi,hucke,lohrey,eric.noeth}@eti.uni-siegen.de

analyzed. Prominent grammar-based string compressors are for instance LZ78, RePair, and BISECTION, see [13] for more details. The theoretically best known polynomial time grammar-based compressors produce for an input string s of length N a grammar of size at most $\mathcal{O}(g \cdot \log(N/g))$, where g is the size of a smallest SLP for s [13,26,28,41,42].

Motivated by applications where large tree structured data occur, like XML processing, grammar-based compression has been extended to trees [9,10,29, 36], see [34] 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 such as term rewriting, expression evaluation and tree automata. A tree over a ranked alphabet is uniquely represented by its preorder traversal string (which is also known as the Polish notation). 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 approach is very similar to [8], where unranked unlabelled trees are compressed by SLPs for their balanced parenthesis representations. In [40] this idea is used together with the grammar-based compressor RePair to get a new compressed suffix tree implementation.

In Section 4 we compare the size of SLPs for preorder traversals with two other grammar-based compressed tree representations: the above mentioned SLPs for balanced parenthesis representations from [8] and (ii) tree straight-line programs (TSLPs) [10,23,29,36]. The latter directly generalize string SLPs to trees using context-free tree grammars that produce a single tree. 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. In [18] it is shown that every tree of size N over a fixed ranked alphabet can be produced by a TSLP of size $\mathcal{O}(\frac{N}{\log N})$ and there exist trees of size N for which any TSLP has size $\Omega(\frac{N}{\log N})$. A grammar-based tree compressor based on TSLPs with an approximation ratio of $\mathcal{O}(\log N)$ was presented in [29]. In [10], it was shown that from a given TSLP 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 (Theorem 4) 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 $\Omega(2^{n/2})$. We also match this lower bound by an upper bound: Given an SLP of size m for the traversal of a tree t of height h and maximal rank r, one can efficiently construct a TSLP for t of size $\mathcal{O}(m \cdot h \cdot r)$ (Theorem 5). Finally, we 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 $\Omega(2^{n/2})$ (Theorem 6). Hence, SLPs for preorder traversals can be exponentially more succinct than SLPs for balanced parenthesis representations. It remains open, whether the opposite behavior is possible as well.

We also study algorithmic problems for trees that are encoded by SLPs. We extend some of the results from [8] 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 the kth 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 (Theorem 7). For a couple of other problems (computation of the height and depth of a node, computation of the Horton-Strahler number, 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 [11], and the NC^1 upper bound was generalized to every fixed tree automaton [31]. If we compress the preorder traversal by an SLP, the problem is still solvable in polynomial time for Boolean expressions (Theorem 13), but there is a fixed tree automaton where the evaluation problem becomes PSPACE-complete (Theorems 18 and 19). Concerning the PSPACE lower bound we prove a stronger statement: There exists a finite semiring for which the evaluation problem for SLP-compressed trees is PSPACE-complete (Theorem 19).

A short version of this paper appeared in [19].

Related work on tree compression. There are 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 [7] can be seen as a variation of TSLPs for unranked trees. Recently, in [22] it was shown that for every tree of size N with σ many node labels, the top DAG has size $\mathcal{O}(\frac{N \cdot \log \log_\sigma N}{\log_\sigma N})$, which improved the bound from [7]. An extension of TSLPs to higher order tree grammars was proposed in [30].

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 time allows efficient querying (ideally in time $\mathcal{O}(1)$) of the data structure. For unlabelled unranked trees of size N there exist representations with $2N + o(N)$ bits that support navigation and some other tree queries in time $\mathcal{O}(1)$ [6, 24, 25, 39]. This

result has been extended to labelled trees, where $(\log \sigma) \cdot N + 2N + o(N)$ bits suffice when σ is the number of node labels [16].

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. Given two strings $u, v \in \Sigma^*$, 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 $1 \leq i \leq \min\{|u|, |v|\}$.

2.1 Complexity classes

We assume familiarity with the basic classes from complexity theory, in particular P, NP and PSPACE. The following definitions are only needed in Section 5.3.3. The counting class $\#\mathsf{P}$ contains all functions $f : \Sigma^* \to \mathbb{N}$ for which there exists a nondeterministic polynomial time machine M such that for every $x \in \Sigma^*$, $f(x)$ is the number of accepting computation paths of M on input x. The class PP (probabilistic polynomial time) contains all problems A for which there exists a nondeterministic polynomial time machine M such that for every input x: $x \in A$ if and only if more than half of all computation paths of M on input x are accepting. By a famous result of Toda [44], the class $\mathsf{P}^{\mathsf{PP}} = \mathsf{P}^{\#\mathsf{P}}$ (i.e., the class of all languages that can be decided in deterministic polynomial time with the help of an oracle from PP) contains the whole polynomial time hierarchy. Hence, if a problem is PP-hard, then this can be seen as a strong indication that the problem does not belong to the polynomial time hierarchy (otherwise the polynomial time hierarchy would collapse).

The levels of the *counting hierarchy* C_i^p $(i \geq 0)$ are inductively defined as follows: $\mathsf{C}_0^p = \mathsf{P}$ and $\mathsf{C}_{i+1}^p = \mathsf{PP}^{\mathsf{C}_i^p}$ (the set of languages accepted by a PP-machine as above with an oracle from C_i^p) for all $i \geq 0$. Let $\mathsf{CH} = \bigcup_{i \geq 0} \mathsf{C}_i^p$ be the counting hierarchy. It is not difficult to show that $\mathsf{CH} \subseteq \mathsf{PSPACE}$, and most complexity theorists conjecture that $\mathsf{CH} \subsetneq \mathsf{PSPACE}$. Hence, if a problem belongs to the counting hierarchy, then this can be seen as an indication that the problem is probably not PSPACE-complete. The counting hierarchy can also be seen as an exponentially blown-up version of the circuit complexity class $\mathsf{DLOGTIME}$-uniform TC^0. This is the class of all languages that can be decided with a constant-depth polynomial-size circuit family of unbounded fan-in that in addition to normal Boolean gates may also use threshold gates. $\mathsf{DLOGTIME}$-uniformity means that one can compute in time $\mathcal{O}(\log n)$ (i) the type of a given gate of the nth circuit, and (ii) whether two given gates of the nth circuit are connected by a wire. Here, gates of the nth circuit are encoded by bit string of length $\mathcal{O}(\log n)$. More details on the counting hierarchy (resp., circuit complexity) can be found in [4] (resp., [45]).

2.2 Trees

A *ranked alphabet* $\mathcal{F}$ is a finite set of symbols where every symbol $f \in \mathcal{F}$ has a rank $\mathsf{rank}(f) \in \mathbb{N}$. We assume that $\mathcal{F}$ contains at least one symbol of rank zero. By $\mathcal{F}_n$ we denote the symbols of $\mathcal{F}$ of rank n. 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 $ft_1 \cdots t_n \in \mathcal{T}(\mathcal{F})$.

A string $s \in \mathcal{F}^*$ is called a *fragment* if there exists a non-empty string $x \in \mathcal{F}^+$ such that $sx \in \mathcal{T}(\mathcal{F})$. Note that the empty string ε is a fragment. Intuitively, a fragment is a tree with gaps. The number of gaps of a fragment $s \in \mathcal{F}^+$ is formally defined as the number n of trees $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ such that $st_1 \cdots t_n \in \mathcal{T}(\mathcal{F})$, and is denoted by $\mathsf{gaps}(s)$. The number of gaps of the empty string is defined as 0. The following lemma states that $\mathsf{gaps}(s)$ is indeed well-defined.

Lemma 1 *The following statements hold:*

- *The set $\mathcal{T}(\mathcal{F})$ is prefix-free, i.e. $t \in \mathcal{T}(\mathcal{F})$ and $tv \in \mathcal{T}(\mathcal{F})$ imply $v = \varepsilon$.*
- *If $t \in \mathcal{T}(\mathcal{F})$, then every suffix of t factors uniquely into a concatenation of strings from $\mathcal{T}(\mathcal{F})$.*
- *For every fragment $s \in \mathcal{F}^+$ there is a unique $n \geq 1$ such that $\{x \in \mathcal{F}^* \mid sx \in \mathcal{T}(\mathcal{F})\} = (\mathcal{T}(\mathcal{F}))^n$.*

Since $\mathcal{T}(\mathcal{F})$ is prefix-free we immediately get:

Lemma 2 *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 such that $w = t_1 \cdots t_n s$.*

Definition 1 Let $w \in \mathcal{F}^*$ and let $w = t_1 \cdots t_n s$ as in Lemma 2. We define $\mathsf{ftg}(w) = (n, \mathsf{gaps}(s))$ ("ftg" stands for "full trees and gaps"). Thus, n counts the number of full trees in w and $\mathsf{gaps}(s)$ is the number of trees missing to make the fragment s a tree.

For better readability, we occasionally write a tree $ft_1 \cdots t_n$ with $f \in \mathcal{F}_n$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ as $f(t_1, \ldots, t_n)$, which corresponds to the standard term representation of trees. We also consider trees in their graph-theoretic interpretation: Let $t \in \mathcal{T}(\mathcal{F})$ be a tree. The nodes of t are the positions $1, \ldots, |t|$ in the string t. The root node is 1. Moreover, if t (viewed as a string) 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 seen as a string is simply the preorder traversal of the tree t seen in its standard graph-theoretic interpretation.

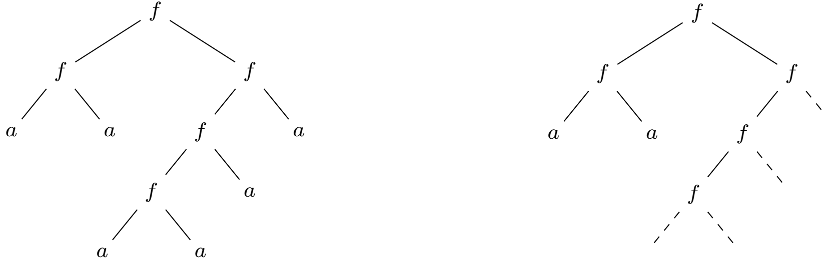

Fig. 1 The tree t from Example 1 and the tree fragment corresponding to the fragment $ffaafff$.

Example 1 Let $t = ffaafffaaaa = f(f(a,a), f(f(f(a,a),a),a))$ be the tree depicted in Figure 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 Figure 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 $\mathsf{ftg}(u) = (2,3)$. The children of node 5 (the third f-labelled node) are 6 and 11.

2.3 Straight-line programs

A *straight-line program*, briefly SLP, is a context-free grammar that produces a single string. Formally, it is a tuple $\mathbb{A} = (\mathcal{N}, \Sigma, P, S)$, where $\mathcal{N}$ is a finite set of nonterminals, Σ is a finite set of terminal symbols ($\Sigma \cap \mathcal{N} = \emptyset$), $S \in \mathcal{N}$ is the start nonterminal, and P is a finite set of productions (or rules) of the form $A \to w$ for $A \in \mathcal{N}$, $w \in (\mathcal{N} \cup \Sigma)^*$ such that:

- For every $A \in \mathcal{N}$, there exists exactly one production of the form $A \to w$, and
- the binary relation $\{(A,B) \in \mathcal{N} \times \mathcal{N} \mid (A \to w) \in P,\ B \text{ occurs in } w\}$ is acyclic.

Every nonterminal $A \in \mathcal{N}$ 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 usually omit the subscript $\mathbb{A}$ when it is clear from the context. The *size* of the SLP $\mathbb{A}$ is $|\mathbb{A}| = \sum_{(A \to w) \in P} |w|$. One can transform an SLP $\mathbb{A} = (\mathcal{N}, \Sigma, P, S)$ which produces a nonempty word in linear time into *Chomsky normal form*, i.e. for each production $(A \to w) \in P$, either $w \in \Sigma$ or $w = BC$ where $B, C \in N$ [33, Proposition 3.8].

For an SLP $\mathbb{A}$ of size n we have $|\mathsf{val}(\mathbb{A})| \in 2^{\mathcal{O}(n)}$, and there exists a family of SLPs $\mathbb{A}_n$ ($n \geq 1$) such that $|\mathbb{A}_n| \in \mathcal{O}(n)$ and $|\mathsf{val}(\mathbb{A})| = 2^n$. Hence, SLPs allow exponential compression.

The following lemma summarizes known results about SLPs which we will use throughout the paper, see e.g. [33].

Lemma 3 *Let $\mathbb{A}$ be an SLP. There are linear time algorithms for the following problems:*

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 ith 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* $\mathsf{val}(\mathbb{A})[i:j]$.

2.4 Tree straight-line programs

We now define tree straight-line programs. Let $\mathcal{F}$ and $\mathcal{V}$ be two disjoint ranked alphabets, where we call elements from $\mathcal{F}$ *terminals* and elements from $\mathcal{V}$ *nonterminals* (or *variables*). Let further $\mathcal{X} = \{x_1, x_2, \dots\}$ be a countably infinite set of *parameters* (disjoint from $\mathcal{F}$ and $\mathcal{V}$), which we treat as symbols of rank zero. In the following we consider trees over $\mathcal{F} \cup \mathcal{V} \cup \mathcal{X}$. The *size* $|t|$ of such a tree t is defined as the number of nodes labelled by a symbol from $\mathcal{F} \cup \mathcal{V}$, i.e. we do not count parameter nodes. A *tree straight-line program* $\mathbb{A}$, or short *TSLP*, is a tuple $\mathbb{A} = (\mathcal{V}, \mathcal{F}, P, S)$, where $\mathcal{V}$ is the set of nonterminals, $\mathcal{F}$ is the set of terminals, $S \in \mathcal{V}_0$ is the start nonterminal and P is a finite set of productions (or rules) of the form $A(x_1, \dots, x_n) \to t$ (which is also briefly written as $A \to t$), where $n \geq 0$, $A \in \mathcal{V}_n$ and $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{V} \cup \{x_1, \dots, x_n\})$ is a tree in which every parameter x_i $(1 \leq i \leq n)$ occurs at most once, such that:

- For every $A \in \mathcal{V}_n$ there is exactly one rule of the form $A(x_1, \dots, x_n) \to t$, and
- 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 $\mathsf{val}_{\mathbb{A}}(A) \in \mathcal{T}(\mathcal{F} \cup \{x_1, \dots, x_n\})$ is derived from every nonterminal $A \in \mathcal{V}_n$ by using the rules as rewriting rules in the usual sense. Thereby, the parameters in the rules are used as variables that can be substituted by arbitrary trees. As for SLPs, we omit the subscript $\mathbb{A}$ when the context is clear. The tree defined by $\mathbb{A}$ is $\mathsf{val}(\mathbb{A}) = \mathsf{val}_{\mathbb{A}}(S)$. Instead of giving a formal definition, we show a derivation of $\mathsf{val}(\mathbb{A})$ from S in an example:

Example 2 Let $\mathbb{A} = (\{S, A, B\}, \{f, a, b\}, F, P, S)$, $S, A \in \mathcal{V}_0, B \in \mathcal{V}_1$, $a, b \in \mathcal{F}_0$, $f \in \mathcal{F}_2$ and

$$P = \{S \to f(A, B(A)),\ A \to B(B(b)),\ B(x_1) \to f(x_1, a)\}.$$

A possible derivation of $\mathsf{val}(\mathbb{A})$ from S is depicted in Figure 2.

The *size* $|\mathbb{A}|$ of a TSLP $\mathbb{A} = (\mathcal{V}, \mathcal{F}, P, S)$ is $|\mathbb{A}| = \sum_{(A \to t) \in P} |t|$. We call a TSLP *monadic* if every nonterminal has rank at most one. One can transform in polynomial time every TSLP $\mathbb{A}$ into a monadic one of size $\mathcal{O}(|\mathbb{A}| \cdot r)$, where

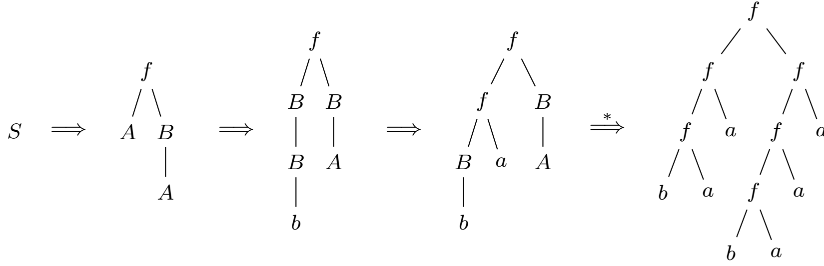

Fig. 2 A derivation of the TSLP from Example 2.

r is the maximal rank of a terminal in $\mathbb{A}$ [37]. TSLPs, where every nonterminal has rank 0, correspond to DAGs (the nodes of the DAG are the nonterminals of the TSLP).

For a TSLP $\mathbb{A}$ of size n we have $|\mathsf{val}(\mathbb{A})| \in 2^{\mathcal{O}(n)}$, and there exists a family of TSLPs $\mathbb{A}_n$ ($n \geq 1$) such that $|\mathbb{A}_n| \in \mathcal{O}(n)$ and $|\mathsf{val}(\mathbb{A})| = 2^n$. Hence, analogously to SLPs, TSLPs allow exponential compression. One can also define *nonlinear* TSLPs where parameters can occur multiple times on right-hand sides; these can achieve doubly exponential compression but have the disadvantage that many algorithmic problems become more difficult, see e.g. [35].

For every word w (resp., tree t) there exists a smallest SLP (resp., TSLP) $\mathbb{A}$. It is known that, unless $\mathsf{P} = \mathsf{NP}$, there is no polynomial time algorithm that finds a smallest SLP (resp., TSLP) for a given word [13] (resp. tree).

3 Checking whether an SLP produces a tree

In this section we show that, given an SLP $\mathbb{A}$ and a ranked alphabet $\mathcal{F}$, we can verify in time linear in $|\mathbb{A}|$, whether $\mathsf{val}(\mathbb{A}) \in \mathcal{T}(\mathcal{F})$. In other words, we present a linear time algorithm for the compressed membership problem for the language $\mathcal{T}(\mathcal{F}) \subseteq \mathcal{F}^*$. We remark that $\mathcal{T}(\mathcal{F})$ is a context-free language, which can be seen by considering the grammar with productions $S \to fS^n$ for all symbols $f \in \mathcal{F}_n$. In general the compressed membership problem for context-free languages can be solved in PSPACE and there exists a deterministic context-free language with a PSPACE-complete compressed membership problem [12,32].

Theorem 1 *Given an SLP $\mathbb{A}$, one can check in time $\mathcal{O}(|\mathbb{A}|)$, whether* $\mathsf{val}(\mathbb{A}) \in \mathcal{T}(\mathcal{F})$.

Proof Let $\mathbb{A} = (\mathcal{N}, \mathcal{F}, P, S)$ be in Chomsky normal form and let $A \in \mathcal{N}$. Due to Lemma 2, we know that $\mathsf{val}(A)$ is the concatenation of trees and a (possibly empty) fragment. Define $\mathsf{ftg}(A) := \mathsf{ftg}(\mathsf{val}(A))$. Then $\mathsf{val}(\mathbb{A}) \in \mathcal{T}(\mathcal{F})$ if and only if $\mathsf{ftg}(S) = (1, 0)$. Hence, it suffices to compute $\mathsf{ftg}(A)$ for all nonterminals

$A \in \mathcal{N}$. We do this bottom-up. If $(A \to f) \in P$ with $f \in \mathcal{F}_n$, then we have

$$\mathsf{ftg}(A) = \begin{cases} (1,0) & \text{if } n = 0 \\ (0,n) & \text{otherwise.} \end{cases}$$

Now consider a nonterminal A with the rule $(A \to BC) \in P$, and let $\mathsf{ftg}(B) = (b_1, b_2)$, $\mathsf{ftg}(C) = (c_1, c_2)$. We claim that

$$\mathsf{ftg}(A) = \begin{cases} (b_1 + c_1 - \max\{1, b_2\} + 1, c_2) & \text{if } b_2 \le c_1 \\ (b_1, c_2 + b_2 - c_1 - \min\{1, c_2\}) & \text{otherwise.} \end{cases}$$

Let $\mathsf{val}(B) = t_1 \cdots t_{b_1} s$ and $\mathsf{val}(C) = t'_1 \cdots t'_{c_1} s'$, where $t_1, \ldots, t_{b_1}, t'_1, \ldots, t'_{c_1} \in \mathcal{T}(\mathcal{F})$ and s (resp., s') is a fragment with $\mathsf{gaps}(s) = b_2$ (resp., $\mathsf{gaps}(s') = c_2$). We distinguish two cases:

Case $b_2 \le c_1$: If $b_2 \ge 1$, then the string $s t'_1 \cdots t'_{b_2}$ is a tree, and thus $\mathsf{val}(A)$ contains $b_1 + 1 + (c_1 - b_2)$ full trees and the fragment s' with c_2 many gaps. On the other hand, if $b_2 = 0$, then $\mathsf{val}(A)$ contains $b_1 + c_1$ many full trees.

Case $b_2 > c_1$: The trees $t'_1, \ldots, t'_{c_1}$ fill c_1 many gaps of s, and if $s' \neq \varepsilon$, then the fragment s' fills one more gap, while creating another c_2 gaps. In total there are $b_2 - (c_1 + 1) + c_2$ gaps if $c_2 > 0$ and $b_2 - c_1$ gaps if $c_2 = 0$. $\square$

Example 3 Let $\mathcal{F} = \{a, f\}$ with $\mathsf{rank}(a) = 0$, $\mathsf{rank}(f) = 2$. Consider the SLP $\mathbb{A}$ with start nonterminal S, terminal rules $F \to f$, $A \to a$ and the remaining rules

$$S \to GH,\ H \to EA,\ G \to CD,\ E \to BD,\ D \to CB,\ C \to FF,\ B \to AA.$$

We want to know whether $\mathsf{val}(\mathbb{A}) \in \mathcal{T}(\mathcal{F})$. From the computations described in Theorem 1, it follows that $\mathsf{ftg}(A) = (1,0)$, $\mathsf{ftg}(F) = (0,2)$, $\mathsf{ftg}(B) = (2,0)$, $\mathsf{ftg}(C) = (0,3)$, $\mathsf{ftg}(D) = (0,1)$, $\mathsf{ftg}(E) = (2,1)$, $\mathsf{ftg}(G) = (0,3)$, $\mathsf{ftg}(H) = (3,0)$ and $\mathsf{ftg}(S) = (1,0)$. Hence, the SLP $\mathbb{A}$ produces a tree $t \in \mathcal{T}(\mathcal{F})$ (which is $ffffaaaaffaaa$).

4 SLPs for traversals versus other grammar-based tree representations

In this section, we compare the worst-case size of SLPs for traversals with the following two grammar-based tree representations:

- TSLPs, and
- SLPs for balanced parenthesis sequences [8].

4.1 SLPs for traversals versus TSLPs

By combining results from [10] and [37] one can show:

Theorem 2 ([10]) *From a given TSLP $\mathbb{A}$ one can compute in polynomial time an SLP $\mathbb{B}$ of size $\mathcal{O}(|\mathbb{A}| \cdot r)$ with $\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{B})$, where r is the maximal rank of a label occurring in $\mathsf{val}(\mathbb{A})$.*

Proof Let $\mathbb{A}$ be a TSLP. By [37] we can transform $\mathbb{A}$ in polynomial time into an equivalent monadic TSLP $\mathbb{A}'$ of size $\mathcal{O}(|\mathbb{A}| \cdot r)$. Then, one can use the construction from [10, proof of Theorem 3] in order to transform $\mathbb{A}'$ into an SLP for the preorder traversal of $\mathsf{val}(\mathbb{A}')$. Since $\mathbb{A}'$ is monadic, it is easy to see that the construction from [10] only involves a constant blow-up. □

Thus, for a binary tree t (where $r = 2$) a smallest SLP for t is only by a constant factor larger than a smallest TSLP for t.

In this section we will discuss the other direction, i.e. transforming an SLP into a TSLP. Let a be a symbol of rank 0 and let f_n be a symbol of rank n for each $n \in \mathbb{N}$. Now let t_n be the tree $f_n a^n$ and consider the family of trees $(t_n)_{n\in\mathbb{N}}$ with unbounded rank. The size of the smallest TSLP for t_n is $n+1$, whereas the size of the smallest SLP for t_n is in $\mathcal{O}(\log n)$. It is less obvious that such an exponential gap can also be realized with trees of bounded rank. In the following we construct a family of binary trees $(t_n)_{n\in\mathbb{N}}$ where a smallest TSLP for t_n is exponentially larger than the size of a smallest SLP for t_n. Afterwards we show that it is always possible to transform an SLP $\mathbb{A}$ for t into a TSLP of size $\mathcal{O}(|\mathbb{A}| \cdot h \cdot r)$ for t, where h is the height of t and r is the maximal rank of a label occurring in t.

4.1.1 Worst-case comparison of SLPs and TSLPs

We use the following result from [5] for the previously mentioned worst-case construction of a family of binary trees:

Theorem 3 (Thm. 2 from [5]) *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 $\Omega(2^{n/2})$.*[1]

For two given 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. See Figure 3 for an illustration.

[1] Actually, in [5] the result is not stated for the convolution $u_n \otimes v_n$, but for the literal shuffle of u_n and v_n which is $u_n[1]v_n[1]u_n[2]v_n[2] \cdots u_n[m]v_n[m]$. But this makes no difference, since the sizes of the smallest SLPs for the convolution and literal shuffle, respectively, of two words differ only by multiplicative constants.

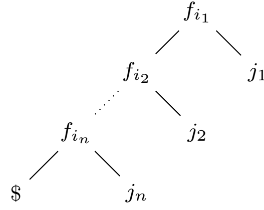

Fig. 3 The comb tree $t(u,v)$ for $u = i_1 \cdots i_n$ and $v = j_1 \cdots j_n$

Theorem 4 *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 Let us fix an n and let u_n and v_n be the aforementioned strings from Theorem 3. Let $|u_n| = |v_n| = m$. Consider the comb tree $t_n := t(u_n, v_n)$. Note that $t_n = f_{i_1} \cdots f_{i_m} \$ \operatorname{rev}(v_n)$, where $u_n = i_1 \cdots i_m$. By Theorem 3 there exist SLPs of size $n^{\mathcal{O}(1)}$ for u_n and v_n, and these SLPs easily yield an SLP of size $n^{\mathcal{O}(1)}$ for t_n.

Next, we show that a TSLP $\mathbb{A}$ for t_n yields an SLP of size $\mathcal{O}(|\mathbb{A}|)$ for the convolution $u_n \otimes v_n$. Since a smallest SLP for $u_n \otimes v_n$ has size $\Omega(2^{n/2})$ by Theorem 3, the same bound must hold for the size of a smallest TSLP for t_n.

Let $\mathbb{A}$ be a TSLP for t_n. By [37] we can transform $\mathbb{A}$ into a monadic TSLP $\mathbb{A}'$ for t_n of size $\mathcal{O}(|\mathbb{A}|)$. We transform the TSLP $\mathbb{A}'$ into an SLP of the same size for $u_n \otimes v_n$. We can assume that every nonterminal except for the start nonterminal S occurs in a right-hand side and every nonterminal occurs in the derivation starting from S. At first we delete all rules of the form $A \to j$ ($j \in \{0,1\}$) and replace the occurrences of A by j in all right-hand sides; this does not increase the size of the TSLP. Now every nonterminal $A \neq S$ of rank 0 derives to a subtree of t_n that contains the unique \$-leaf of t_n. Hence, t_n contains a unique occurrence of the subtree $\mathsf{val}(A)$. This implies that A occurs exactly once in a right hand side. We can therefore without size increase replace this occurrence of A by the right-hand side of A. After this step, S is the only rank-0 nonterminal in the TSLP. With the same argument, we can also eliminate rank-1 nonterminals that derive to a tree containing the unique leaf \$. After this step, every rank-1 nonterminal $A(x)$ derives a tree of the form $g_1(g_2(\ldots(g_k(x, j_k)\ldots), j_2), j_1)$ ($g_i \in \{f_0, f_1\}$ and $j_i \in \{0,1\}$).

Now, if a right-hand side contains a subtree $f_i(s_1, s_2)$, then s_2 must be either 0 or 1. Similarly, for every occurrence of $i \in \{0,1\}$ in a right-hand side, the parent node of that occurrence must be either labelled with f_0 or f_1 (note that the parent node exists and cannot be a nonterminal, since such a nonterminal would have rank two). Therefore we can obtain an SLP for $u_n \otimes v_n$ by replacing every production $A(x) \to t(x)$ by $A \to \lambda(t(x))$, where $\lambda(t(x))$ is the string obtained inductively by $\lambda(x) = \varepsilon$, $\lambda(B(s(x)) = B\lambda(s(x))$ for nonterminals B, and $\lambda(f_i(s(x), j)) = (i,j)\lambda(s(x))$. The production for S must be of the form $S \to t(\$)$ for a term $t(x)$ and we replace it by $S \to \lambda(t(x))\$$.
□

4.1.2 Conversion of SLPs to TSLPs

Note that the height of the tree t_n in Theorem 4 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 5 *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 Without loss of generality we assume that $\mathbb{A}$ is in Chomsky normal form. For every nonterminal A of $\mathbb{A}$ with $\mathsf{ftg}(A) = (a_1, a_2)$ we introduce a_1 nonterminals $A_1, \dots, A_{a_1}$ of rank 0 (these produce one tree each) and, if $a_2 > 0$, one nonterminal A' of rank a_2 for the fragment encoded by A. 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 $n \geq 1$. Now consider a rule of the form $A \to BC$ with $\mathsf{ftg}(B) = (b_1, b_2)$ and $\mathsf{ftg}(C) = (c_1, c_2)$.

Case 1: If $b_2 = 0$ we add the following rules to $\mathbb{B}$:

$$\begin{aligned} A_i &\to B_i && \text{for } 1 \leq i \leq b_1 \\ A_{b_1+i} &\to C_i && \text{for } 1 \leq i \leq c_1 \\ A'(x_1, \dots, x_{c_2}) &\to C'(x_1, \dots, x_{c_2}) && \text{if } c_2 > 0. \end{aligned}$$

Case 2: If $0 < b_2 \leq c_1$ we add the following rules to $\mathbb{B}$:

$$\begin{aligned} A_i &\to B_i && \text{for } 1 \leq i \leq b_1 \\ A_{b_1+1} &\to B'(C_1, \dots, C_{b_2}) && \\ A_{b_1+1+i} &\to C_{b_2+i} && \text{for } 1 \leq i \leq c_1 - b_2 \\ A'(x_1, \dots, x_{c_2}) &\to C'(x_1, \dots, x_{c_2}) && \text{if } c_2 > 0. \end{aligned}$$

Case 3: If $b_2 > c_1$ we add the following rules to $\mathbb{B}$, where $d = b_2 - c_1$:

$$\begin{aligned} A_i &\to B_i && \text{for } 1 \leq i \leq b_1 \\ A'(x_1, \dots, x_d) &\to B'(C_1, \dots, C_{c_1}, x_1, \dots, x_d) && \text{if } c_2 = 0 \\ A'(x_1, \dots, x_{c_2+d-1}) &\to && \\ & B'(C_1, \dots, C_{c_1}, C'(x_1, \dots, x_{c_2}), x_{c_2+1}, \dots, x_{c_2+d-1}) && \text{if } c_2 > 0. \end{aligned}$$

Chain productions, where the right-hand side consists of a single nonterminal, can be eliminated without size increase. Then, only one of the above productions remains and its size is bounded by $c_1 + 2$ (recall that we do not count parameters). Recall that c_1 is the number of complete trees produced by C. It therefore suffices to show that the number of complete trees of a factor s of t is bounded by $h \cdot r$, where h is the height of t and r is the maximal rank of a label in t. Assume that $s = t[i : j] = t_1 \cdots t_n s'$, where $t_i \in \mathcal{T}(\mathcal{F})$ and s' is a fragment. Let k be the lowest common ancestor of i and j. If $k = i$ (i.e., i

is an ancestor of j) then either $s = t_1$ or $s = s'$. Otherwise, the root of every tree t_l ($1 \leq l \leq n$) is a child of a node on the path from i to k. The length of the path from i to k is bounded by h, hence $n \leq h \cdot r$. $\square$

Example 4 Let $\mathcal{F} = \{a, f\}$ with $\mathsf{rank}(a) = 0$, $\mathsf{rank}(f) = 2$. Consider the tree $t = ffffaaaaffaaa$, which is produced by the SLP $\mathbb{A}$ with start nonterminal S, terminal rules $F \to f$, $A \to a$ and the remaining rules

$$S \to GH,\ H \to EA,\ G \to CD,\ E \to BD,\ D \to CB,\ C \to FF,\ B \to AA.$$

This SLP was considered in Example 3, where the ftg-values of the nonterminals were computed. From the construction in the proof of Theorem 5 we obtain an TSLP for t with the following rules (without eliminating chain productions):

$$\begin{gathered} A_1 \to a,\ F'(x_1,x_2) \to f(x_1,x_2),\ B_1 \to A_1,\ B_2 \to A_1, \\ C'(x_1,x_2,x_3) \to F'(F'(x_1,x_2),x_3),\ D'(x_1) \to C'(B_1,B_2,x_1), \\ E_1 \to B_1,\ E_2 \to B_2,\ E'(x_1) \to D'(x_1),\ G'(x_1,x_2,x_3) \to C'(D'(x_1),x_2,x_3), \\ H_1 \to E_1,\ H_2 \to E_2,\ H_3 \to E'(A_1),\ S \to G'(H_1,H_2,H_3). \end{gathered}$$

4.2 SLPs for traversals versus balanced parenthesis sequences

Balanced parenthesis sequences are widely used as a succinct representation of ordered unranked unlabeled trees [39]. 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 $\mathsf{bp}(t) = \texttt{()}$. If the root of t has n children in which the subtrees $t_1, \ldots, t_n$ are rooted (from left to right), then $\mathsf{bp}(t) = \texttt{(}\mathsf{bp}(t_1)\cdots\mathsf{bp}(t_n)\texttt{)}$. Hence, a tree with n nodes is represented by $2n$ bits, which is optimal in the information theoretic sense. On the other hand, an unlabelled full binary tree t (i.e., a tree where every non-leaf node has exactly two children) of size n can be represented with n bits by viewing t as a ranked tree over $\mathcal{F} = \{a, f\}$, where f has rank two and a has rank zero.

Theorem 6 *For every $n > 0$ there exists a full binary tree t_n 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 $\Omega(2^{n/2})$.*

Proof Let us fix an n and let $u_n, v_n \in \{0,1\}^*$ be the strings from Theorem 3. Let $|u_n| = |v_n| = m$. We define t_n by

$$t_n = \varphi_1(\mathrm{rev}(u_n))\, a\, \varphi_2(v_n)$$

where $\varphi_1, \varphi_2 : \{0,1\}^* \to \{a, f\}^*$ are the homomorphisms defined as follows:

$$\begin{aligned} \varphi_1(0) &= f & \qquad \varphi_2(0) &= a \\ \varphi_1(1) &= faf & \qquad \varphi_2(1) &= faa \end{aligned}$$

Fig. 4 Example tree for the proof of Theorem 6

It is easy to see that t_n is indeed a tree (note that the string $\varphi_2(v_n)$ is a sequence of m many trees). From the SLPs for u_n and v_n we obtain an SLP for t_n of size polynomial in n. It remains to show that the smallest SLP for $\mathsf{bp}(t_n)$ has size $\Omega(2^{n/2})$. To do so, we show that from an SLP for $\mathsf{bp}(t_n)$ we can obtain with a linear size increase an SLP for the convolution of u_n and v_n. In fact, we show the following claim:

Claim. The convolution $u_n \otimes v_n$ can be obtained from a suffix of $\mathsf{bp}(t_n)$ by a fixed rational transformation (i.e., a deterministic finite automaton that outputs along every transition a finite word over some output alphabet).

This claim proves the theorem using the following two facts:

- An SLP for a suffix of a string $\mathsf{val}(\mathbb{A})$ (for an SLP $\mathbb{A}$) can be produced by an SLP of size $\mathcal{O}(|\mathbb{A}|)$ by point 4 of Lemma 3.
- For every fixed rational transformation ρ, an SLP for $\rho(\mathsf{val}(\mathbb{A}))$ can be produced by an SLP of size $\mathcal{O}(|\mathbb{A}|)$ [5, Theorem 1] (the $\mathcal{O}$-constant depends on the rational transformation).

To see why the above claim holds, it is the best to look at an example. Assume that $u_n = 10100$ and $v_n = 10010$. Hence, we have

$$t_n = \varphi_1(\mathrm{rev}(u_n))\, a\, \varphi_2(v_n) \ = \ f\ f\ faf\ f\ faf\ a\ faa\ a\ a\ faa\ a.$$

This tree is shown in Figure 4. We have

$$\mathsf{bp}(t_n) = \underset{0}{\texttt{(}}\ \underset{0}{\texttt{(}}\ \underbrace{\texttt{(()(}}_{1}\ \underset{0}{\texttt{(}}\ \underbrace{\texttt{(()(}}_{1}\ \texttt{()}\ \underbrace{\texttt{(()()))}}_{(1,1)}\ \underbrace{\texttt{())}}_{(0,0)}\ \underbrace{\texttt{()))}}_{(1,0)}\ \underbrace{\texttt{(()()))}}_{(0,1)}\ \underbrace{\texttt{())}}_{(0,0)}\ .$$

Indeed, $\mathsf{bp}(t_n)$ starts with an encoding of the string $\mathrm{rev}(u_n)$ (here 00101) via the correspondence $0 \mathrel{\widehat{=}} \texttt{(}$ and $1 \mathrel{\widehat{=}} \texttt{(()(}$, followed by $\texttt{()}$ (which encodes

the single a between $\varphi_1(\mathrm{rev}(u_n))$ and $\varphi_2(v_n)$ in t_n), followed by the desired encoding of the convolution $u_n \otimes v_n$. The latter is encoded by the following correspondence:

$$\begin{aligned}(0,0) &\mathrel{\widehat{=}} \texttt{())}\\(1,0) &\mathrel{\widehat{=}} \texttt{()))}\\(0,1) &\mathrel{\widehat{=}} \texttt{(()()))}\\(1,1) &\mathrel{\widehat{=}} \texttt{(()())))}.\end{aligned}$$

So, a 0 in the second component is encoded by `()`, which corresponds to the tree a. A 1 in the second component is encoded by `(()())`, which corresponds to the tree faa. A 0 (resp., 1) in the first component is encoded by one (resp., two) closing parenthesis. Let $\varphi : \{(0,0),(0,1),(1,0),(1,1)\}^* \to \{\texttt{())},\texttt{()))},\texttt{(()()))},\texttt{(()())))}\}^*$ be the mapping defined by the above correspondence. Note that the strings `())`, `()))`, `(()()))`, `(()())))` form a code, i.e., φ is injective and hence bijective. Therefore, the inverse φ^{-1} exists. Moreover, φ^{-1} can be computed by a deterministic rational transducer. The transducer has to buffer at most eight brackets in order to output the next symbol from $\{(0,0),(1,0),(0,1)\}$. This shows the above claim. □

5 Algorithmic problems on SLP-compressed trees

In this section we study the complexity of several basic algorithmic problems on trees that are represented by SLPs.

5.1 Efficient tree operations

In [8] it is shown that for a given SLP $\mathbb{A}$ of size n that produces the balanced parenthesis representation of an unranked tree t of size N, one can produce in time $\mathcal{O}(n)$ a data structure of size $\mathcal{O}(n)$ that supports navigation as well as other important tree queries (e.g. lowest common ancestors queries) 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 [7,22] 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 $1,\dots,|t|$ in the string t.

Theorem 7 *Given an SLP $\mathbb{A}$ of size n for a tree $t \in \mathcal{T}(\mathcal{F})$ 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 $\mathcal{O}(\log N) \le \mathcal{O}(n)$ on a word RAM, where $i,j,k \in \mathbb{N}$ with $1 \le i,j \le N$ are given in binary notation:*

(a) Compute the parent node of node $i > 1$ in t.
(b) Compute the kth child of node i in t, if it exists.

(c) Compute the number k such that $i > 1$ is the kth child of its parent node.
(d) Compute the size of the subtree rooted at node i.
(e) Compute the lowest common ancestor of nodes i and j in t.

Proof In [8], it is shown that for an SLP $\mathbb{A}$ of size n that produces a well-parenthesized string $w \in \{(,)\}^*$ of length 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 $\mathcal{O}(\log N)$ on a word RAM, where $1 \le k, j \le N$ are given in binary notation and $b \in \{(,)\}$:

- Compute the number of positions $1 \le i \le k$ such that $w[i] = b$ ($\mathsf{rank}_b(k)$).
- Compute the position of the kth occurrence of b in w if it exists ($\mathsf{select}_b(k)$).
- Compute the position of the matching closing (resp., opening) parenthesis for an opening (resp., closing) parenthesis at position k ($\mathsf{findclose}(k)$ and $\mathsf{findopen}(k)$).
- Compute the left-most position $i \in [k, j]$ having the smallest excess value in the interval $[k, j]$, where the excess value at a position i is $\mathsf{rank}_((i) - \mathsf{rank}_)(i)$ ($\mathsf{rmqi}(k, j)$).

Let us now take an SLP $\mathbb{A}$ of size n for a tree $t \in \mathcal{T}(\mathcal{F})$ of size N and let s be the corresponding unlabelled tree. In [6], the DFUDS-representation (DFUDS for depth-first-unary-degree-sequence) of s is defined as follows: Walk over the tree in preorder and write down for every node with d children the string $(^d)$ (d opening parenthesis followed by a closing parenthesis). Finally put an additional opening parenthesis at the beginning of the resulting string, which yields a well-parenthesized string. For instance, for the tree $g(f(a, a), a, h(a))$ we obtain the DFUDS-representation `((((()(())))()).` Clearly, from the SLP $\mathbb{A}$ we can produce an SLP $\mathbb{B}$ for the DFUDS-representation of the tree s: Simply replace in right-hand sides every occurrence of a symbol f of rank d by $(^d)$, and add an opening parenthesis in front of the right-hand side of the start nonterminal.

The starting position of the encoding of a node $i \in \{1, \ldots, N\}$ in the DFUDS-representation can be found as $\mathsf{select}_)(i-1) + 1$ for $i > 1$, and for $i = 1$ it is 2. Vice versa if k is the starting position of the encoding of a node in the DFUDS-representation, then the preorder number of that node is $\mathsf{rank}_)(k-1) + 1$.

In [6,25], it is shown that the tree navigation operations from the theorem can be implemented on the DFUDS-representation using a constant number of rank, select, $\mathsf{findclose}(k)$, $\mathsf{findopen}(k)$ and rmqi-operations. Together with the above mentioned results from [8] this shows the theorem. $\square$

The data structure of [8] allows to compute the depth and height of a given tree node in time $\mathcal{O}(\log N)$ as well. It is not clear to us, whether this result can be extended to our setting as well. In [25] it is shown that the depth of a given node can be computed in constant time on the DFUDS-representation. But this uses an extra data structure, and it is not clear whether this extra data structure can be adapted so that it works for an SLP-compressed DFUDS-representation. On the other hand, in Section 5.3, we show that the height

and depth of a given node of an SLP-compressed tree can be computed in polynomial time.

For a full binary (unlabelled) tree t let $\mathsf{dfuds}(t)$ be the DFUDS-representation of t. Then, an SLP $\mathbb{A}$ of size n for the tree t can be transformed into an SLP $\mathbb{B}$ of size $3n+1$ for the string $\mathsf{dfuds}(t)$: as in the above proof one has to replace every occurrence of f (resp., a) in a right-hand side of $\mathbb{A}$ by `(()` (resp., `)`) and add a `(` in front of the right-hand side of the start nonterminal. Together with Theorem 6 this shows the following corollary:

Theorem 8 *For every $n > 0$ there exists a full binary tree t_n such that the size of a smallest SLP for $\mathsf{dfuds}(t_n)$ 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 tree family where the opposite situation arises, i.e., where the size of a smallest SLP for the balanced parenthesis representation grows polynomially with n but the size of a smallest SLP for the DFUDS-representation grows exponentially with n.

5.2 Pattern matching

In contrast to navigation problems, simple pattern matching problems become intractable for SLP-compressed trees. The *pattern matching problem for SLP-compressed trees* can be formalized as follows: Given a tree $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{X})$, called the *pattern*, where every parameter $x \in \mathcal{X}$ 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 the tree $\sigma(x)$. Note that the pattern is given in uncompressed form. If the tree t is given by a TSLP, the corresponding problem can be solved in polynomial time. This can be deduced from [43] (where the more general problem with a TSLP-compressed pattern tree s is solved in polynomial time) or [37] (where it is shown that a given tree automaton can be evaluated on a TSLP-compressed tree in polynomial time).

Theorem 9 *The pattern matching problem for SLP-compressed trees is* NP*-complete. Moreover,* NP*-hardness holds for a fixed pattern of the form $f(x, a)$*

Proof The problem is contained in NP because one can guess a node $i \in \{1, \ldots, |t|\}$ and verify whether the subtree of t rooted in i matches the pattern s. The verification is possible in polynomial time by comparing all relevant symbols using Theorem 7.

By [33, Theorem 3.13] it is NP-complete to decide for given SLPs $\mathbb{A}, \mathbb{B}$ over $\{0, 1\}$ with $|\mathsf{val}(\mathbb{A})| = |\mathsf{val}(\mathbb{B})|$ whether there exists a position i such that $\mathsf{val}(\mathbb{A})[i] = \mathsf{val}(\mathbb{B})[i] = 1$. This question can be reduced to the pattern matching problem with a fixed pattern. One can compute in polynomial time from $\mathbb{A}$ and $\mathbb{B}$ an SLP $\mathbb{T}$ for the comb tree $t(\mathsf{val}(\mathbb{A}), \mathsf{val}(\mathbb{B}))$. There exists a position i such that $\mathsf{val}(\mathbb{A})[i] = \mathsf{val}(\mathbb{B})[i] = 1$ if and only if the pattern $f_1(x, 1)$ occurs in $t(\mathsf{val}(\mathbb{A}), \mathsf{val}(\mathbb{B}))$. $\square$

Let us remark that pattern matching for SLP-compressed *strings* (i.e., the question whether for given SLPs $\mathbb{T}$ (the text SLP) and $\mathbb{P}$ (the pattern SLP) there exist words u and v such that $\mathsf{val}(\mathbb{T}) = u\,\mathsf{val}(\mathbb{P})\,v$) can be solved in polynomial time. The currently best known algorithm by Jeż [27] has a running time of $O((|\mathbb{T}| + |\mathbb{P}|) \log |\mathsf{val}(\mathbb{P})|)$ under the assumption that $|\mathsf{val}(\mathbb{P})|$ can be stored in a single machine word; otherwise an additional factor $\log(|\mathbb{T}| + |\mathbb{P}|)$ goes in.

5.3 Tree evaluation problems

In this section we study the complexity of various tree evaluation problems for SLP-compressed input trees. In some cases, these evaluation problems will be more difficult for SLP-compressed trees than TSLP-compressed trees. An example for this situation is the evaluation problem for tree automata: For TSLP-compressed input trees, this problem can be solved in polynomial [37] time, while it becomes PSPACE-complete for SLP-compressed input trees; see Section 5.3.4. On the other hand, in Section 5.3.2 we will present several evaluation problems that can be solved in polynomial time for SLP-compressed input trees (and hence by Theorem 2 also for TSLP-compressed input trees). Examples are the computation of the height of a tree and the evaluation of Boolean expression trees. In these cases, our polynomial time algorithms for SLPs are more involved than the corresponding algorithms for TSLPs. Let us consider for instance the computation of the height of a tree. 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 framework 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 of constants of a ranked alphabet $\mathcal{F}$ to be infinite, i.e. $\mathcal{F}_0 \subseteq \mathbb{Z}$.

Definition 2 Let $\mathcal{D} \subseteq \mathbb{Z}$ be a (possibly infinite) domain 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 function 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 $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$.

Definition 3 The *$\mathcal{I}$-evaluation problem for SLP-compressed trees* is the following problem: Given an SLP $\mathbb{A}$ over $\mathcal{F}$ with $\mathsf{val}(\mathbb{A}) \in \mathcal{T}(\mathcal{F})$, compute $\mathsf{val}(\mathbb{A})^{\mathcal{I}}$.

5.3.1 Reduction to caterpillar trees

In this section, we reduce the $\mathcal{I}$-evaluation problem 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 2 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 $s^{\mathcal{I}} = t_1^{\mathcal{I}} \cdots t_n^{\mathcal{I}} u^{\mathcal{I}}$ with $t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}} \in \mathcal{D}$. Since u is a fragment, the string $u^{\mathcal{I}}$ is the fragment of a caterpillar tree (briefly, caterpillar fragment in the following).

Example 5 Let $\mathcal{F} = \{0, 1, 2, +, \times\}$ with the standard interpretation on integers ($+$ and $\times$ are considered as binary operators). Consider

$$s = 0, 2, +, 2, +, +, \times, 2, +, 2, 1, +, \times$$

(commas are added for better readability). Since $+, 2, 1$ evaluates to 3, and $\times, 2, 3$ evaluates to 6, we have $s^{\mathcal{I}} = 0, 2, +, 2, +, +, 6, +, \times$.

Our reduction to caterpillar trees only works for interpretations that satisfy a certain growth condition. We say that an interpretation $\mathcal{I}$ is *polynomially bounded*, if there exist constants $\alpha, \beta \geq 0$ such that for every tree $t \in \mathcal{T}(\mathcal{F})$ (we denote the absolute value of an integer by z by $\mathsf{abs}(z)$ instead of $|z|$ in order to not get confused with the size $|t|$ of a tree),

$$\mathsf{abs}(t^{\mathcal{I}}) \leq \left(\beta \cdot |t| + \sum_{i \in L} \mathsf{abs}(t[i]) \right)^{\alpha}$$

where $L \subseteq \{1, \ldots, |t|\}$ is the set of leaves of t. The purpose of this definition is to ensure that for every SLP $\mathbb{A}$ with $\mathsf{val}(\mathbb{A}) \in \mathcal{T}(\mathcal{F})$, both the length of the binary encoding of $\mathsf{val}(\mathbb{A})^{\mathcal{I}}$ and the integer constants that appear in $\mathbb{A}$ are polynomially bounded in $|\mathbb{A}|$.

Theorem 10 *Let $\mathcal{I}$ be a polynomially bounded interpretation. Then the $\mathcal{I}$-evaluation problem for SLP-compressed trees is polynomial time Turing-reducible to the $\mathcal{I}$-evaluation problem 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* $\mathbb{A} = (\mathcal{N}, \Sigma, P, S)$ is an SLP where P may also contain rules of the form $A \to B[i : j]$ where $A, B \in \mathcal{N}$ and $i, j \geq 0$. Here we let $\mathsf{val}(A) = \mathsf{val}(B)[i : j]$. It is known [20] (see also [33]) that a given composition system can be transformed in polynomial time into an SLP with the same value. One can also allow mixed rules $A \to X_1 \cdots X_n$ where each X_i is either a terminal, a nonterminal or an expression of the form $B[i : j]$, which clearly can be eliminated in polynomial time.

Let $\mathbb{A} = (\mathcal{N}, \mathcal{F}, P, S)$ be the input SLP in Chomsky normal form. We use the notation $\mathsf{ftg}(A) = \mathsf{ftg}(\mathsf{val}(A))$ as in the proof of Theorem 1. We will compute a composition system where for each nonterminal $A \in \mathcal{N}$ there are nonterminals A_1 and A_2 in the composition system 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 $\mathsf{ftg}(A) = (n, \mathsf{gaps}(s))$). 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 computation is straightforward for rules of the form $A \to f$ with $A \in \mathcal{N}$ 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 $A \in \mathcal{N}$ with the rule $A \to BC$ we make a case distinction depending on $\mathsf{ftg}(B) = (b_1, b_2)$ and $\mathsf{ftg}(C) = (c_1, c_2)$.

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

$$A_1 \to B_1\, t_{\mathrm{new}}^{\mathcal{I}}\, C_1[b_2 + 1 : c_1]$$
$$A_2 \to C_2$$

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 $A_2 \to B_2C_2$. 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 5. 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

Fig. 5 An example for the case $b_2 > c_1$ in the proof of Theorem 10. In the left fragment we insert the trees a, b, a, a and the fragment faf. The latter yield, together with a part of the fragment, a new tree t_{cat}.

$r = |\mathsf{val}(B_2)| + c_1 + 1$ (the position of the first $\diamond$ in t'). Let q be the parent node of r, which can be computed in polynomial time by Theorem 7. Using Lemma 3 we compute the position p 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 $A_2 \to B_2C_1C_2$ to the composition system. Otherwise p is the first symbol of the largest factor from $\mathcal{T}(\mathcal{F})$ described above. Using Theorem 7(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 $t_{\text{cat}} = (\mathsf{val}(B_2)\,\mathsf{val}(C_1))[p : p']$ 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 $t_{\text{cat}}^{\mathcal{I}}$. We then add the rules

$$\begin{aligned} A_1 &\to B_1, \\ A' &\to B_2C_1, \text{ and} \\ A_2 &\to A'[: p-1]\, t_{\text{cat}}^{\mathcal{I}}\, A'[p'+1:]\, C_2 \end{aligned}$$

to the composition system. This completes the proof. □

5.3.2 Polynomial time solvable evaluation problems

Next, we present several applications of Theorem 10. We start with the height of a tree.

Theorem 11 *The height of a tree $t \in \mathcal{T}(\mathcal{F})$ given by an SLP $\mathbb{A}$ is computable 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 Theorem 10 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 in this caterpillar tree, arbitrary natural numbers may occur at leaf positions.

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 largest (i.e., deepest) node such that d is the label of a child of node v_d (in particular, v_d is not a leaf). Since t is a caterpillar tree, the node v_d is well-defined. 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 Theorem 7 we compute the parent nodes of those two nodes in t and take the maximum (i.e., the deeper one) of both. This node 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 larger than zero occurs (called k above). Let t_i be the subtree rooted at v_{d_i}. Then $t_m^{\mathcal{I}} = d_m + 1$. We now 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 we can ignore all constants that appear in the interval $[v_{d_i}+1, v_{d_{i+1}}-1]$ except for d_i. More precisely, assume that $a = t_{i+1}^{\mathcal{I}}$ and let b be the number of occurrences of symbols of rank at least one in the interval $[v_{d_i}+1, v_{d_{i+1}}-1]$. Also this number can be computed in polynomial time by Lemma 3. Then the value of $t_i^{\mathcal{I}}$ is $\max\{a+b+1, d_i+1\}$. Finally, using the same argument, we can compute $t^{\mathcal{I}}$ from $t_1^{\mathcal{I}}$. □

Corollary 1 *Given an SLP $\mathbb{A}$ for a tree t and a node $1 \le i \le |t|$ one can compute the depth of i in t in polynomial time.*

Proof We can write t as $t = uvw$, where $|u| = i-1$ and v is the subtree of t rooted at node i. We can compute $|v|$ in polynomial time by Theorem 7. This allows to compute in polynomial time an SLP for the tree $uh^{|t|}aw$. Here, h has rank one and a has rank zero. Then the depth of i in t is $\mathrm{height}(uh^{|t|}aw) - |t|$. □

An interesting parameter of a tree t is its *Horton-Strahler number* or *Strahler number*, see [15] for a recent survey. 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 $f^{\mathcal{I}}(a_1,\ldots,a_n) = a+1$. The Strahler number was first defined in hydrology, but also has many applications in computer science [15], e.g. to calculate the minimum number of registers required to evaluate an arithmetic expression [17].

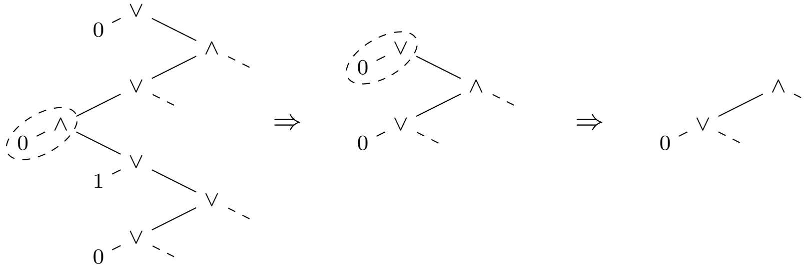

Fig. 6 An example for step 1 in the proof of Theorem 13. In the first image we find the expression $\wedge 0$, hence we remove the remaining suffix. The expression $\vee 0$ can also be removed without changing the final truth value.

Theorem 12 *Given an SLP $\mathbb{A}$ for a tree t, one can compute the Strahler number of t in polynomial time.*

Proof Note that the interpretation $\mathcal{I}$ above is very similar to the one from the proof of Theorem 11. The only difference is that the uniqueness of the maximum among the children of a node also affects the evaluation. Therefore the proof of Theorem 11 must be slightly modified by considering for each $d \in \mathbb{N}$ occurring in t the two deepest leaves in t labelled with d (or the unique leaf labelled by d if d occurs exactly once). Let i and j be the parents of those two leaves ($i \geq j$) and let t_i (resp., t_j) be the subtree of t rooted at i (resp., j). The nodes i and j can be computed in polynomial time as in the proof of Theorem 11. We have $t_i^{\mathcal{I}} \geq d$, and therefore $t_j^{\mathcal{I}} = d + 1$. This implies that any further occurrence of d that is higher up in the tree has no influence on the evaluation process. The rest of the argument is similar to the proof of Theorem 11. $\square$

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

Proof Let $\mathbb{A}$ be an SLP over $\{\wedge, \vee, 0, 1\}$ such that $\mathsf{val}(\mathbb{A})$ is a caterpillar tree. Define a *left caterpillar tree* to be a tree of the form uv, where $u \in \{\wedge, \vee\}^*$, $v \in \{0, 1\}^*$ and $|v| = |u|+1$. That means that the main branch of the caterpillar tree grows to the left. The evaluation of $\mathsf{val}(\mathbb{A})$ is done in two steps. In a first step, we compute in polynomial time from $\mathbb{A}$ a new SLP $\mathbb{B}$ such that $\mathbb{B}$ is a left caterpillar tree and $\mathsf{val}(\mathbb{A})^{\mathcal{I}} = \mathsf{val}(\mathbb{B})^{\mathcal{I}}$. In a second step, we show how to evaluate a left caterpillar tree. We can assume that $\mathsf{val}(\mathbb{A})$ is neither 0 or 1.

Step 1. (See Figure 6 for an illustration of step 1.) Since $\mathsf{val}(\mathbb{A})$ is a caterpillar tree, we have $\mathsf{val}(\mathbb{A}) = uv$ with $u \in \{\wedge, \vee, \wedge 0, \wedge 1, \vee 0, \vee 1\}^* \cdot \{\wedge, \vee\}$, $v \in \{0, 1\}^*$ and $|v|$ is 1 plus the number of occurrences of the symbols $\wedge, \vee$ in u that are

not followed by 0 or 1 in u. We can compute bottom-up the length of the maximal suffix of $\mathsf{val}(\mathbb{A})$ from $\{0,1\}^*$ in polynomial time. Hence, by Lemma 3 we can compute in polynomial time SLPs $\mathbb{A}_1$ and $\mathbb{A}_2$ such that $\mathsf{val}(\mathbb{A}_1) = u$ and $\mathsf{val}(\mathbb{A}_2) = v$.

We will show how to eliminate all occurrences of the patterns $\wedge 0$, $\wedge 1$, $\vee 0$, $\vee 1$ from $\mathsf{val}(\mathbb{A}_1)$. For this, it is technically easier to replace every occurrence of $\circ a$ by a new symbol $\circ_a$, where $\circ \in \{\wedge, \vee\}$ and $a \in \{0,1\}$. Let $\varphi : \{\wedge, \vee, \wedge 0, \wedge 1, \vee 0, \vee 1\}^* \to \{\wedge, \vee, \wedge_0, \wedge_1, \vee_0, \vee_1\}^*$ be the mapping that replaces every occurrence of $\circ a$ by the new symbol $\circ_a$ ($\circ \in \{\wedge, \vee\}$, $a \in \{0,1\}$). This mapping is a rational transformation. Hence, using [5, Theorem 1], we can compute in polynomial time an SLP $\mathbb{B}_1$ for $\varphi(\mathsf{val}(\mathbb{A}_1))$. We now compute, using Lemma 3, the position i in $\mathsf{val}(\mathbb{B}_1)$ of the first occurrence of a symbol from $\{\wedge_0, \vee_1\}$. Next, we compute an SLP $\mathbb{C}_1$ for the prefix $\mathsf{val}(\mathbb{B}_1)[:i-1]$, i.e., we cut off the suffix starting in position i. Moreover, we compute the number j of occurrences of symbols from $\{\wedge, \vee\}$ in the suffix $\mathsf{val}(\mathbb{B}_1)[i:]$ and compute an SLP $\mathbb{B}_2$ for the string $0\,\mathsf{val}(\mathbb{A}_2)[j+2:]$ in case $\mathsf{val}(\mathbb{B}_1)[i] = \wedge_0$ and $1\,\mathsf{val}(\mathbb{A}_2)[j+2:]$ in case $\mathsf{val}(\mathbb{B}_1)[i] = \vee_1$. Then $\mathsf{val}(\mathbb{A})$ evaluates to the same truth value as $\varphi^{-1}(\mathsf{val}(\mathbb{C}_1))\,\mathsf{val}(\mathbb{B}_2)$. The reason for this is that $\varphi^{-1}(\mathsf{val}(\mathbb{B}_1)[i:])\,\mathsf{val}(\mathbb{A}_2)[:j+1]$ is a tree which evaluates to 0 (resp., 1) if $\mathsf{val}(\mathbb{B}_1)[i] = \wedge_0$ (resp., $\mathsf{val}(\mathbb{B}_1)[i] = \vee_1$), because $0 \wedge x = 0$ (resp., $1 \vee x = 1$).

Note that $\varphi^{-1}(\mathsf{val}(\mathbb{C}_1))\,\mathsf{val}(\mathbb{B}_2)$ is a caterpillar tree, where $\mathsf{val}(\mathbb{B}_2) \in \{0,1\}^*$ and $\mathsf{val}(\mathbb{C}_1) \in \{\wedge, \vee, \wedge_1, \vee_0\}^*$. Since $1 \wedge x = x$ (resp., $0 \vee x = x$), we can delete in the string $\mathsf{val}(\mathbb{C}_1)$ all occurrences of the symbols $\wedge_1$ and $\vee_0$ without changing the final truth value. Let $\mathbb{D}_1$ be an SLP for the resulting string, which is easy to compute from $\mathbb{C}_1$. Then $\mathsf{val}(\mathbb{D}_1)\,\mathsf{val}(\mathbb{B}_2)$ is indeed a left caterpillar tree.

Step 2. To evaluate a left caterpillar tree let $\mathbb{A}_1$ and $\mathbb{A}_2$ be two SLPs where $\mathsf{val}(\mathbb{A}_1) \in \{\wedge, \vee\}^*$, $\mathsf{val}(\mathbb{A}_2) \in \{0,1\}^*$, and $|\mathsf{val}(\mathbb{A}_2)| = |\mathsf{val}(\mathbb{A}_1)| + 1$. Let $\varphi : \{\wedge, \vee\}^* \to \{0,1\}^*$ be the homomorphism with $\varphi(\wedge) = 1$ and $\varphi(\vee) = 0$. Using binary search, we compute the largest position i such that the reversed length-i suffix of $\mathsf{val}(\mathbb{A}_2)$ is equal to the length-i prefix of $\varphi(\mathsf{val}(\mathbb{A}_1))$. If $i = |\mathsf{val}(\mathbb{A}_1)|$, then the value of $\mathsf{val}(\mathbb{A}_1)\,\mathsf{val}(\mathbb{A}_2)$ is the first symbol of $\mathsf{val}(\mathbb{A}_2)$. Otherwise, the value of $\mathsf{val}(\mathbb{A}_1)\,\mathsf{val}(\mathbb{A}_2)$ is 0 (resp., 1) if $\mathsf{val}(\mathbb{A}_1)[i+1] = \wedge$ (resp., $\mathsf{val}(\mathbb{A}_1)[i+1] = \vee$). □

Corollary 2 *If the interpretation $\mathcal{I}$ is such that $(\mathcal{D}, \wedge^{\mathcal{I}}, \vee^{\mathcal{I}})$ is a finite distributive lattice, then the $\mathcal{I}$-evaluation problem for SLP-compressed trees can be solved in polynomial time.*

Proof By Birkhoff's representation theorem, every finite distributive lattice is isomorphic to a lattice of finite sets, where the join (resp., meet) operation is set union (resp., intersection). This lattice embeds into a finite power of $(\{0,1\}, \wedge, \vee)$. □

5.3.3 Difficult arithmetical evaluation problems

Assume that $\mathcal{I}$ is the interpretation that assigns to the symbols $+$ and $\times$ their standard meaning over the integers. Note that this interpretation is not

polynomially bounded. For instance, for the tree $t_n = \times^n(2)^{n+1}$ we have $t_n^{\mathcal{I}} = 2^{n+1}$. Hence, if a tree t is given by an SLP $\mathbb{A}$, then the number of bits of $t^{\mathcal{I}}$ can be exponential in the size of $\mathbb{A}$. Therefore, we cannot write down the number $t^{\mathcal{I}}$ in polynomial time. The same problem arises already for numbers that are given by arithmetic circuits (circuits over $+$ and $\times$).

In [3] it was shown that the problem of computing the kth bit (k is given in binary notation) of the number to which a given arithmetic circuit evaluates to belongs to the counting hierarchy. An arithmetic circuit can be seen as a DAG that unfolds to an expression tree. Dags correspond to TSLPs where all nonterminals have rank 0. Vice versa, it was shown in [18] that a TSLP $\mathbb{A}$ over $+$ and $\times$ can be transformed in logspace into an arithmetic circuit that evaluates to $\mathsf{val}(\mathbb{A})^{\mathcal{I}}$. This transformation holds for any semiring. Thus, over semirings, the evaluation problems for TSLPs and circuits (i.e., DAGs) have the same complexity. In particular, the problem of computing the kth bit of the output value of a TSLP-represented arithmetic expression belongs to the counting hierarchy. Here, we show that this result even holds for arithmetic expressions that are given by SLPs:

Theorem 14 *The problem of computing for a given binary encoded number k and an SLP $\mathbb{A}$ over $\{+,\times\}\cup\mathbb{Z}$ the kth bit of $\mathsf{val}(\mathbb{A})^{\mathcal{I}}$ belongs to the counting hierarchy.*

Proof We follow the strategy from [3, proof of Thm. 4.1]. Let $\mathbb{A}$ be the input SLP for the tree t and let $M = \mathcal{I}(t)$. Then $M \le 2^{2^n}$ where $n = |\mathbb{A}|$ (this follows since the expression t has size at most 2^n and the value computed by an expression of size m is at most 2^m). Let P_n be the set of all prime numbers in the range $[2, 2^{2n}]$ (note that $2^{2n} \ge \log^2 M$). Then $\prod_{p\in P_n} p > M$. Also note that each prime $p \in P_n$ has at most $2n$ bits in its binary representation. We first show that the language

$$L = \{(\mathbb{A}, p, j) \mid \mathbb{A} \text{ is an SLP for a tree, } n = |\mathbb{A}|,\ p \in P_n,\ 1 \le j \le 2n, \text{ the } j^{\text{th}} \text{ bit of } \mathsf{val}(\mathbb{A})^{\mathcal{I}} \bmod p \text{ is } 1\}$$

belongs to the counting hierarchy. The rest of proof then follows the argument in [3]: Using the $\mathsf{DLOGTIME}$-uniform TC^0-circuit family from [21] for transforming a number from its Chinese remainder representation into its binary representation one defines a TC^0-circuit of size $2^{\mathcal{O}(n)}$ that has input gates $x(p,j)$ (where $n = |\mathbb{A}|$, $p \in P_n$, $1 \le j \le 2n$). If we set $x(p,j)$ to true iff $(\mathbb{A}, p, j) \in L$ (this means that the input gates $x(p,j)$ receive the Chinese remainder representation of $\mathsf{val}(\mathbb{A})^{\mathcal{I}}$), then the circuit outputs correctly the (exponentially many) bits of the binary representation of $\mathsf{val}(\mathbb{A})^{\mathcal{I}}$. Then, as in [3, proof of Thm. 4.1], one shows by induction on the depth of a gate that the problem whether a given gate of that circuit (the gate is specified by a bit string of length $\mathcal{O}(n)$) evaluates to true is in the counting hierarchy, where the level in the counting hierarchy depends on the level of the gate in the circuit.[2]

[2] Let us explain the differences to [3, proof of Thm. 4.1]: In [3], the arithmetic expression is given by a circuit instead of an SLP. This simplifies the proof, because if we replace in the

Hence we have to show that L belongs to the counting hierarchy. Let $\mathbb{A}$ be an SLP for a tree t, $n = |\mathbb{A}|$, $p \in P_n$, and $1 \leq j \leq 2n$. By Theorem 10 it suffices to consider the case that t is a caterpillar tree t; the polynomial time Turing reduction in Theorem 10 increases the level in the counting hierarchy by one. Also note that we use a uniform version of Theorem 10, where the interpretation (addition and multiplication in $\mathbb{Z}_p$) is part of the input. This is not a problem, since the prime number p has at most $2n$ bits, so all values that can appear only need $2n$ bits.

Let m be the number of operators in t, i.e., the total number of occurrences of the symbols $+$ and $\times$ in $\mathsf{val}(\mathbb{A})$. Note that m can be exponentially large in $|\mathbb{A}|$, but its binary representation can be computed in polynomial time by Lemma 3 (point 2). We now define a matrix of numbers $x^t_{i,j} \in \mathbb{Z}_p$ ($i, j \in [1, m+1]$) such that

$$t^{\mathcal{I}} = \sum_{i=1}^{m+1} \prod_{j=1}^{m+1} x^t_{i,j}.$$

Moreover, we will show that given $\mathbb{A}$ and binary encoded numbers $i, j \in [1, m+1]$, the binary encoding of $x^t_{i,j}$ (which consists of at most $2n$ bits) can be computed in polynomial time.

We define the numbers $x^t_{i,j}$ inductively over the structure of the caterpillar tree t. For the caterpillar tree $t = a$ (with $a \in \mathbb{Z}_p$) we set $x^t_{1,1} = a$. Now assume that $t = f(a, s)$ or $t = f(s, a)$ for an operator $f \in \{+, \times\}$, a caterpillar tree s with $m - 1$ operators, and $a \in \mathbb{Z}_p$. In the case $t = f(s, a)$ we assume that $m - 1 \geq 1$; this avoids ambiguities in case $t = f(a, b)$ for $a, b \in \mathbb{Z}_p$. Assume that the numbers $x^s_{i,j}$ are already defined for $i, j \in [1, m]$. If $f = +$, then we set:

$$\begin{aligned}
x^t_{1,1} &= a \\
x^t_{1,i} &= 1 \text{ for } i \in [2, m+1] \\
x^t_{i,1} &= 1 \text{ for } i \in [2, m+1] \\
x^t_{i,j} &= x^s_{i-1,j-1} \text{ for } i, j \in [2, m+1]
\end{aligned}$$

We get

$$\sum_{i=1}^{m+1} \prod_{j=1}^{m+1} x^t_{i,j} = a + \sum_{i=2}^{m+1} \prod_{j=2}^{m+1} x^s_{i-1,j-1} = a + \sum_{i=1}^{m} \prod_{j=1}^{m} x^s_{i,j} = a + s^{\mathcal{I}} = t^{\mathcal{I}}.$$

If $f = \times$, then we set:

$$\begin{aligned}
x^t_{1,i} &= 0 \text{ for } i \in [1, m+1] \\
x^t_{i,1} &= a \text{ for } i \in [2, m+1] \\
x^t_{i,j} &= x^s_{i-1,j-1} \text{ for } i, j \in [2, m+1]
\end{aligned}$$

above language L the SLP $\mathbb{A}$ by a circuit, then we can decide the language L in polynomial time (we only have to evaluate a circuit modulo a prime number with polynomially many bits). In our situation, we can only show that L belongs to a certain level of the counting hierarchy. But this suffices to prove the theorem, only the level in the counting hierarchy increases by the number of levels in which the set L sits.

We get

$$\sum_{i=1}^{m+1}\prod_{j=1}^{m+1} x^t_{i,j} = \sum_{i=2}^{m+1} a \cdot \prod_{j=2}^{m+1} x^s_{i-1,j-1} = a \cdot \sum_{i=1}^{m}\prod_{j=1}^{m} x^s_{i,j} = a \cdot s^{\mathcal{I}} = t^{\mathcal{I}}.$$

We now show that the binary encodings of the numbers $x^t_{i,j}$ can be computed in polynomial time (given $\mathbb{A}, i, j$). For this let us introduce some notations: For our caterpillar tree $t = \mathsf{val}(\mathbb{A})$ (which contains m occurrences of operators) and $i \in [1, m]$, $j \in [1, m+1]$ we define inductively $\mathrm{op}(t, i) \in \{+, \times\}$ and $\mathrm{operand}(t, j) \in \mathbb{Z}_p$ as follows:

– If $t = a \in \mathbb{Z}_p$, then let $\mathrm{operand}(t, 1) = a$ (note that in this case we have $m = 0$, hence the $\mathrm{op}(t, i)$ do not exist).
– If $t = f(a, s)$ or ($t = f(s, a)$ and $m \geq 2$) with $a \in \mathbb{Z}_p$, then we set $\mathrm{op}(t, 1) = f$, $\mathrm{op}(t, i) = \mathrm{op}(s, i-1)$ for $i \in [2, m]$, $\mathrm{operand}(t, 1) = a$, and $\mathrm{operand}(t, j) = \mathrm{operand}(s, j-1)$ for $j \in [2, m+1]$.

In other words: $\mathrm{op}(t, i)$ is the ith operator in t, and $\mathrm{operand}(t, j)$ is the unique argument from $\mathbb{Z}_p$ of the jth operator in t (recall that t is a caterpillar tree). The mth (and hence last) operator in t has two arguments from $\mathbb{Z}_p$; its left argument is $\mathrm{operand}(t, m)$ and its right argument is $\mathrm{operand}(t, m+1)$. Using these notations, we can compute the numbers $x^t_{i,j}$ by the following case distinction (correctness follows by a straightforward induction):

– $i < j$: If $\mathrm{op}(t, i) = +$ then $x^t_{i,j} = 1$, else $x^t_{i,j} = 0$.
– $i = j$: If $\mathrm{op}(t, i) = +$ then $x^t_{i,j} = \mathrm{operand}(t, j)$, else $x^t_{i,j} = 0$.
– $i > j$: If $\mathrm{op}(t, j) = +$ then $x^t_{i,j} = 1$, else $x^t_{i,j} = \mathrm{operand}(t, j)$.

So, in order to compute the $x^t_{i,j}$ it suffices to compute $\mathrm{op}(t, i)$ and $\mathrm{operand}(t, j)$, given $\mathbb{A}, i, j$. This is possible in polynomial time: The position k of the ith operator in t and $\mathrm{op}(t, i)$ can be computed in polynomial time using point 3 of Lemma 3 (take $\Gamma = \{+, \times\}$). Once the position k is computed, $\mathrm{operand}(t, i)$ can be computed in polynomial time using point (b) of Theorem 7.

Recall that our goal is to compute a specific bit of $\mathsf{val}(\mathbb{A})^{\mathcal{I}} \bmod p$, where $\mathbb{A}$ is an SLP that produces a caterpillar tree, and $p \in [2, 2^{2n}]$ is a prime, where $n = |\mathbb{A}|$. We have to show that this problem belongs to the counting hierarchy. We have shown that

$$\mathsf{val}(\mathbb{A})^{\mathcal{I}} = \sum_{i=1}^{m+1}\prod_{j=1}^{m+1} x^t_{i,j}.$$

where the binary encoding of the number $x^t_{i,j} \in \mathbb{Z}_p$ can be computed in polynomial time, given $\mathbb{A}, i, j$. We now follow again the arguments from [3]. It is known that the binary representation of a sum (resp., product) of n many n-bit numbers can be computed in $\mathsf{DLOGTIME}$-uniform TC^0 [21]. The same holds for the problem of computing a sum (resp., product) of n many numbers from $[0, p-1]$ modulo a given prime number p with $\mathcal{O}(\log n)$ bits (it is actually much easier to argue that the latter problem is in $\mathsf{DLOGTIME}$-uniform

TC^0, see again [21]). Hence, there is a $\mathsf{DLOGTIME}$-uniform TC^0 circuit family $(C_m)_{m\geq 1}$, where the input of C_m consists of bits $x(i,j,k)$ ($i,j \in [1,m]$, $k \in \mathcal{O}(\log m)$) and a prime number p with $\mathcal{O}(\log m)$ bits, such that the following holds: If $x(i,j,k)$ receives the kth bit of a number $x_{i,j} \in \mathbb{Z}_p$, then the circuit outputs $\sum_{i=1}^m \prod_{j=1}^m x_{i,j} \bmod p$. We take the circuit C_{m+1}, where $m \in 2^{\mathcal{O}(n)}$ (recall that $n = |\mathbb{A}|$ and m is the number of operators in $t = \mathsf{val}(\mathbb{A})$). The input gate $x(i,j,k)$ receives the kth bit of the number $x^t_{i,j} \in \mathbb{Z}_p$ defined above. We have shown above that the bits of $x^t_{i,j}$ can be computed in polynomial time. This allows (again in the same way as in [3, proof of Thm. 4.1]) to show that for a given gate number of C_{m+1} one can compute the truth value of the corresponding gate within the counting hierarchy. □

Computing a certain bit of the output number of an arithmetic circuit belongs to $\mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}$ [2] (but no matching lower bound is known). In our situation, the level gets even higher, so we made no effort to compute it.

We can use the technique from the proof of Theorem 14 to show the following related result. Note that a circuit (or DAG) over max and + can be evaluated in polynomial time (simply by computing bottom-up the value of each gate), and by the reduction from [18] the same holds for TSLP-compressed expressions.

Theorem 15 *The problem of evaluating SLP-compressed* $(\{\max,+\}\cup\mathbb{Z})$*-trees over the integers belongs to the counting hierarchy.*

Proof The proof follows the arguments from the proof of Theorem 14. But since the interpretation given by max and + is polynomially bounded, every subtree of an SLP-compressed tree evaluates to an integer that needs only polynomially many bits with respect to the size of the SLP. Hence we do not need the Chinese remainder theorem as in the proof of Theorem 14 and can use Theorem 10 directly. It remains to show that the problem of evaluating SLP-compressed $(\{\max,+\}\cup\mathbb{Z})$-caterpillar trees belongs to the counting hierarchy. For this we follow the same strategy as in the proof of Theorem 14 and define numbers $x^t_{i,j}$ (where $t = \mathsf{val}(\mathbb{A})$ is the input caterpillar tree) such that

$$\mathsf{val}(\mathbb{A})^{\mathcal{I}} = \max_{1\leq i\leq m+1} \sum_{j=1}^{m+1} x^t_{i,j}.$$

Since the sum of n many n-bit numbers as well as the maximum of n many n-bit numbers can be computed in $\mathsf{DLOGTIME}$-uniform TC^0 (the maximum of n many n-bit numbers can be even computed in $\mathsf{DLOGTIME}$-uniform AC^0), one can argue as in the proof of Theorem 14. □

Let us now turn to lower bounds for the problems of evaluating SLP-compressed arithmetic expressions (max-plus or plus-times). For a number $c \in \mathbb{N}$ consider the unary operation $+_c$ on $\mathbb{N}$ with $+_c(z) = z + c$. The evaluation of SLP-compressed $(\{\max,+_c\}\cup\mathbb{N})$-trees is possible in polynomial time analogously to the proof of Theorem 11. The following theorem shows that the general case of SLP-compressed $(\{\max,+\}\cup\mathbb{N})$-trees is more complicated.

Theorem 16 *Evaluating SLP-compressed* $(\{\max, +\} \cup \mathbb{N})$*-trees is* $\#\mathsf{P}$*-hard.*

Proof Let $\mathbb{A}, \mathbb{B}$ be two SLPs over $\{0,1\}$ with $|\mathsf{val}(\mathbb{A})| = |\mathsf{val}(\mathbb{B})|$. We will reduce from the problem of counting the number of occurrences of $(1,1)$ in the convolution $\mathsf{val}(\mathbb{A}) \otimes \mathsf{val}(\mathbb{B}) \in (\{0,1\}^2)^*$, which is known to be $\#\mathsf{P}$-complete by [32]. Let $\rho : \{0,1\}^* \to \{\max, +\}^*$ be the homomorphism defined by $\rho(0) = \max$, $\rho(1) = +$. One can compute in polynomial time from $\mathbb{A}$ and $\mathbb{B}$ an SLP for the tree $\rho(\mathsf{val}(\mathbb{A}))\, 1\, \mathrm{rev}(\mathsf{val}(\mathbb{B}))$. The corresponding tree over $\{\max, +, 0, 1\}$ evaluates to one plus the number of occurrences of $(1,1)$ in the convolution $\mathsf{val}(\mathbb{A}) \otimes \mathsf{val}(\mathbb{B})$. □

In [3] it was shown that the computation of a certain bit of the output value of an arithmetic circuit (over $+$ and $\times$) is $\#\mathsf{P}$-hard. Since a circuit can be seen as a TSLP (where all nonterminals have rank 0), which can be transformed in polynomial time into an SLP for the same tree [10], also the problem of computing a certain bit of $\mathsf{val}(\mathbb{A})^{\mathcal{I}}$ for a given SLP $\mathbb{A}$ is $\#\mathsf{P}$-hard. For the related problem PosSLP of deciding, whether a given arithmetic circuit computes a positive number, no non-trivial lower bound is known. For SLPs, the corresponding problem becomes PP-hard:

Theorem 17 *The problem of deciding whether* $\mathsf{val}(\mathbb{A})^{\mathcal{I}} \geq 0$ *for a given SLP* $\mathbb{A}$ *over* $\{+, \times\} \cup \mathbb{Z}$ *is* PP*-hard.*

Proof By [32], the following problem is PP-complete: Given SLPs $\mathbb{A}, \mathbb{B}$ over $\{0,1\}$ where $|\mathsf{val}(\mathbb{A})| = |\mathsf{val}(\mathbb{B})|$, and a binary encoded number z, is the number of occurrences of $(1,1)$ in the convoluted string $\mathsf{val}(\mathbb{A}) \otimes \mathsf{val}(\mathbb{B})$ at least z? We modify the proof of Theorem 16. Let $\mathbb{A}, \mathbb{B}$ be SLPs over $\{0,1\}$, where $N = |\mathsf{val}(\mathbb{A})| = |\mathsf{val}(\mathbb{B})|$. Pick $n \geq 0$ such that $2^n > 2N$. Let $\rho_A : \{0,1\}^* \to \{+, \times\}^*$ be the homomorphism defined by $\rho_A(0) = +$, $\rho_A(1) = \times$ and $\rho_B : \{0,1\}^* \to \{1,2\}^*$ be the homomorphism defined by $\rho_B(0) = 1$, $\rho_B(1) = 2$. One can compute in polynomial time from $\mathbb{A}$ and $\mathbb{B}$ an SLP for the tree $\rho_A(\mathsf{val}(\mathbb{A}))\,(2^n)\,\rho_B(\mathrm{rev}(\mathsf{val}(\mathbb{B})))$ (here 2^n stands for an SLP that evaluates to 2^n). Let R be the value of the corresponding tree. Note that R is calculated by starting with the value 2^n and applying N additions or multiplications by 1 or 2. The number K of occurrences of $(1,1)$ in the convolution $\mathsf{val}(\mathbb{A}) \otimes \mathsf{val}(\mathbb{B})$ corresponds to the number of multiplications by 2 in the calculation, which can be computed from R: We have

$$2^n \cdot 2^K \leq R \leq (2^n + 2(N-K)) \cdot 2^K \leq (2^n + 2N) \cdot 2^K$$

since R is maximal if $(N-K)$ additions of 2 are followed by K multiplications by 2. Since $2N < 2^n$ we obtain $2^{n+K} \leq R \leq 2^{n+K} + r$ for some $r < 2^{n+K}$. Hence, $K \geq z$, if and only if $R - 2^{n+z} \geq 0$. It is straightforward to compute an SLP which evaluates to $R - 2^{n+z}$. □

5.3.4 Tree automata and finite semirings

Deterministic (bottom-up) tree automata (see [14] for details) can be seen as finite algebras: The domain of the algebra is the set of states, and the operations of the algebra correspond to the transitions of the automaton. Nondeterministic tree automata generalize deterministic tree automata and are defined as follows: A *nondeterministic (bottom-up) tree automaton* $\mathcal{A} = (Q, \mathcal{F}, \Delta, F)$ consists of a finite set of *states* Q, a ranked alphabet $\mathcal{F}$, a set Δ of *transition rules* of the form $f(q_1, \dots, q_n) \to q$ where $f \in \mathcal{F}_n$ and $q_1, \dots, q_n, q \in Q$, and a set of *final states* $F \subseteq Q$. 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. Using the powerset construction, a nondeterministic tree automaton can be transformed into an equivalent deterministic tree automaton.

The uniform membership problem for tree automata asks whether a given tree automaton $\mathcal{A}$ accepts a given tree $t \in \mathcal{T}(\mathcal{F})$. In [31] it was shown that this problem is complete for the class LogCFL, which is the closure of the context-free languages under logspace reductions. The inclusions $\mathsf{LogCFL} \subseteq \mathsf{P}$ and $\mathsf{LogCFL} \subseteq \mathsf{DSPACE}(\log^2(n))$ are well-known. For every fixed tree automaton, the membership problem belongs to NC^1 [31]. If the input tree is given by a TSLP, the uniform membership problem becomes P-complete [37]. For non-linear TSLPs (where a parameter may occur several times in a right-hand side) the uniform membership problem becomes PSPACE-complete, and PSPACE-hardness holds already for a fixed tree automaton [35]. The same complexity bound holds for SLP-compressed trees (which in contrast to non-linear TSLPs only allow exponential compression).

Theorem 18 *Given a tree automaton $\mathcal{A}$ and an SLP $\mathbb{A}$ for a tree $t \in \mathcal{T}(\mathcal{F})$, one can decide in polynomial space whether $\mathcal{A}$ accepts t.*

Proof We use the following lemma from [38]: If a function $f : \Sigma^* \to \Gamma^*$ is PSPACE-computable and $L \subseteq \Gamma^*$ belongs to $\mathsf{NSPACE}(\log^k(n))$ for some constant k, then $f^{-1}(L)$ belongs to PSPACE. Given an SLP $\mathbb{A}$ for the tree $t = \mathsf{val}(\mathbb{A})$, one can compute the tree t by a PSPACE-transducer by computing the symbol $t[i]$ for every position $i \in \{1, \dots, |t|\}$. The current position can be stored in polynomial space and every query can be performed in polynomial time by Lemma 3. As remarked above the uniform membership problem for explicitly given trees can be solved in $\mathsf{DSPACE}(\log^2(n))$. □

In the short version [19] of this paper we proved that there exists a fixed tree automaton $\mathcal{A}$ such that it is PSPACE-complete to check whether a given SLP-compressed tree is accepted by $\mathcal{A}$. Here we strengthen this result slightly. Note that a fixed finite algebraic structure $(A, f_1, \dots, f_k)$ (where A is a finite set and every f_i is an operation of a certain arity on A) can be simulated by a (deterministic) tree automaton: A is the set of states and there is a transition $f(a_1, \dots, a_n) \to a$ if $f(a_1, \dots, a_n) = a$. Hence, the next theorem indeed strengthens the above mentioned result from [19]. Recall that a (non-commutative) semiring is a structure $(S, +, \times)$, where $(S, +)$ is a commutative

monoid with identity element 0, $(S, \times)$ is a monoid, $0 \times s = s \times 0 = 0$ for all $s \in S$, and $\times$ left and right distributes over $+$.

Theorem 19 *There is a finite (non-commutative) semiring $\mathcal{S} = (S, +, \times)$ together with a finite subset $F \subseteq S$ such that the following problem is* PSPACE*-complete: Given an SLP-compressed $(\{+, \times\} \cup S)$-tree t, does t evaluate in $\mathcal{S}$ to an element of F?*

Proof Let us fix the morphism $\rho : (\{0,1\} \times \{0,1\})^* \to \{0,1\}^*$ with $\rho(0,0) = \rho(0,1) = \varepsilon$ and $\rho(1,x) = x$ for $x \in \{0,1\}$. In [32] the existence of a regular language $K \subseteq \{0,1\}^*$ such that the following problem is PSPACE-complete was shown: Given two SLPs $\mathbb{A}$ and $\mathbb{B}$ over $\{0,1\}$ with $|\mathsf{val}(\mathbb{A})| = |\mathsf{val}(\mathbb{B})|$, is $\rho(\mathsf{val}(\mathbb{A}) \otimes \mathsf{val}(\mathbb{B})) \in K$?

Let $L = aK$, where $a \notin \{0,1\}$ is a new symbol. Let M be the syntactic monoid of the regular language L (which is a finite monoid) and $h : \{0,1,a\}^* \to M$ be the syntactic morphism. This means that there is a subset $T \subseteq M$ such that for all $w \in \{0,1,a\}^*$: $w \in L$ if and only if $h(w) \in T$. Our semiring $\mathcal{S}$ will be the power semiring of M, which is denoted by $\mathcal{P}(M)$. Its elements are the subsets of M, semiring addition is defined by the union of sets, and semiring multiplication (which we denote with $\times$) is defined by the pointwise product of sets, i.e., $U \times V = \{uv \mid u \in U, v \in V\}$ for all $U, V \subseteq M$. We define the homomorphism $g : \{0,1\}^* \to \{\cup, \times\}^*$ by $g(0) = \cup$ and $g(1) = \times$.

Let us now take two SLPs $\mathbb{A}$ and $\mathbb{B}$ over $\{0,1\}$ with $|\mathsf{val}(\mathbb{A})| = |\mathsf{val}(\mathbb{B})|$. Let $s = \mathsf{val}(\mathbb{A})$ and $t = a_1 a_2 \cdots a_n = \mathsf{val}(\mathbb{B})$. From $\mathbb{A}$ and $\mathbb{B}$ we can construct an SLP $\mathbb{C}$ over the alphabet $\{\cup, \times\} \cup M$ for the string

$$g(\mathrm{rev}(s))h(a)h(a_1)h(a_2)\cdots h(a_n).$$

By identifying an element $m \in M$ with the element $\{m\}$ of the power semiring $\mathcal{P}(M)$, we can view $\mathsf{val}(\mathbb{C})$ as an SLP-compressed caterpillar tree over the power semiring. Let $U \subseteq M$ be the subset to which $\mathsf{val}(\mathbb{C})$ evaluates. This subset can be computed as follows: Let $P \subseteq \{1, \ldots, |s|\}$ be the set of all positions p in the string s such that $s[p] = 0$ (i.e., $g(s)[p] = \cup$). Then we have

$$U = \{h(a\rho(s \otimes t))\} \cup \{h(t[p]\rho(s[p+1:] \otimes t[p+1:])) \mid p \in P\} \tag{1}$$

Instead of giving a formal proof of this, let us present an example. Let $s = 1101001$ and $t = a_1a_2a_3a_4a_5a_6a_7$ with $a_i \in \{0,1\}$. Figure 7 shows the caterpillar tree $g(\mathrm{rev}(s))h(a)h(a_1)h(a_2)\cdots h(a_n)$. We get

$$U = \{h(aa_1a_2a_4a_7), h(a_3a_4a_7), h(a_5a_7), h(a_6a_7)\},$$

which is also the set we get from (1).

We claim that $\rho(\mathsf{val}(\mathbb{A}) \otimes \mathsf{val}(\mathbb{B})) \in K$ if and only if $U \cap T \neq \emptyset$, which proves the theorem. From (1) it follows that $U \cap T \neq \emptyset$ if and only if

$$\{a\rho(s \otimes t)\} \cup \{t[p]\rho(s[p+1:] \otimes t[p+1:]) \mid p \in P\} \cap L \neq \emptyset.$$

Since $L = aK$ with $K \subseteq \{0,1\}^*$, this is equivalent to $\rho(s \otimes t) \in L$, which concludes the proof. $\square$

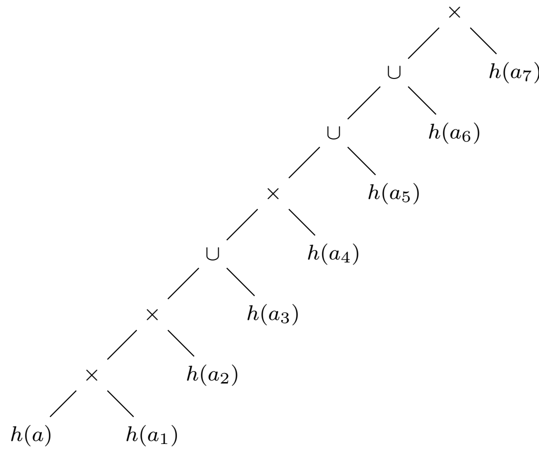

Fig. 7 The caterpillar tree from the proof of Theorem 19. The bit string $s = 1101001$ is translated into $g(\mathrm{rev}(s)) = \times \cup \cup \times \cup \times \times$.

Theorems 18 and 19 imply that there exists a fixed tree automaton for which the membership problem for SLP-compressed trees is PSPACE-complete. This result 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 [37], 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 #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).

6 Further research

We conjecture that in practice, grammar-based tree compression based on SLPs leads to faster compression and better compression ratios compared to grammar-based tree compression based on TSLPs, and we plan to substantiate this conjecture with experiments on real tree data. The theoretical results from Section 4 indicate that SLPs may achieve better compression ratios than TSLPs. Moreover, grammar-based string compression can be implemented without pointer structures, whereas all grammar-based tree compressors (that construct TSLPs) we are aware of work with pointer structures for trees, and a string-encoded tree (e.g. an XML document) must be first transformed into a pointer structure. Moreover, we believe that SLPs can be encoded more succinctly than TSLPs (for instance, we do not have to store the ranks of nonterminals).

References

1. T. Akutsu. A bisection algorithm for grammar-based compression of ordered trees. *Information Processing Letters*, 110(18-19):815–820, 2010.
2. E. Allender, N. Balaji, and S. Datta. Low-depth uniform threshold circuits and the bit-complexity of straight line programs. In *Proceedings of MFCS 2014, Part II*, volume 8635 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2014.
3. E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. Bro Miltersen. On the complexity of numerical analysis. *SIAM Journal on Computing*, 38(5):1987–2006, 2009.
4. E. Allender and K. W. Wagner. Counting hierarchies: Polynomial time and constant depth circuits. *Bulletin of the EATCS*, 40:182–194, 1990.
5. A. Bertoni, C. Choffrut, and R. Radicioni. Literal shuffle of compressed words. In *Proceedings of IFIP TCS 2008*, volume 273 of *IFIP*, pages 87–100. Springer, 2008.
6. 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.
7. P. Bille, I. L. Gørtz, G. M. Landau, and O. Weimann. Tree compression with top trees. *Information and Computation*, 243: 166–177, 2015.
8. P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. R. Satti, and O. Weimann. Random access to grammar-compressed strings and trees. *SIAM Journal on Computing*, 44(3)513–539:, 2015.
9. M. Bousquet-Mélou, M. Lohrey, S. Maneth, and E. Noeth. XML compression via DAGs. *Theory of Computing Systems*, 2014.
10. G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Information Systems*, 33(4–5):456–474, 2008.
11. S. R. Buss. The Boolean formula value problem is in ALOGTIME. In *Proceedings of STOC 1987*, pages 123–131. ACM Press, 1987.
12. H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57(2):200–212, 1998.
13. M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.
14. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://tata.gforge.inria.fr/`.
15. J. Esparza, M. Luttenberger, and M. Schlund. A brief history of strahler numbers. In *Proceedings of LATA 2014*, volume 8370 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2014.
16. P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *Journal of the ACM*, 57(1), 2009.
17. P. Flajolet, J.-C. Raoult, and J. Vuillemin. The number of registers required for evaluating arithmetic expressions. *Theoretical Computer Science*, 9:99–125, 1979.
18. M. Ganardi, D. Hucke, A. Jèz, M. Lohrey, and E. Noeth. Constructing small tree grammars and small circuits for formulas. Technical report, arXiv.org, 2014. `http://arxiv.org/abs/1407.4286`.
19. M. Ganardi, D. Hucke, M. Lohrey, E. Noeth. Tree compression using string grammars. In *Proceedings of LATIN 2016*, volume 9644 of *Lecture Notes in Computer Science*, pages 590–604. Springer, 2016.
20. C. Hagenah. *Gleichungen mit regulären Randbedingungen über freien Gruppen*. PhD thesis, University of Stuttgart, Institut für Informatik, 2000.
21. W. Hesse, E. Allender, and D. A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002.
22. L. Hübschle-Schneider and R. Raman. Tree compression with top trees revisited. In *Proceedings of SEA 2015*, volume 9125 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2015.
23. D. Hucke, M. Lohrey, and E. Noeth. Constructing small tree grammars and small circuits for formulas. In *Proceedings of FSTTCS 2014*, volume 29 of *LIPIcs*, pages 457–468. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.

24. G. Jacobson. Space-efficient static trees and graphs. In *Proceedings of FOCS 1989*, pages 549–554. IEEE Computer Society, 1989.
25. J. Jansson, K. Sadakane, and W-K. Sung. Ultra-succinct representation of ordered trees with applications. *Journal of Computer and System Sciences*, 78(2):619–631, 2012.
26. A. Jeż. Approximation of grammar-based compression via recompression. *Theoretical Computer Science*, 592:115–134, 2015.
27. A. Jeż. Faster fully compressed pattern matching by recompression. *ACM Transactions on Algorithms*, 11(3):20:1–20:43, 2015.
28. A. Jeż. A really simple approximation of smallest grammar. *Theoretical Computer Science*, 616:141–150, 2016.
29. A. Jeż and M. Lohrey. Approximation of smallest linear tree grammars. In *Procedings of STACS 2014*, volume 25 of *LIPIcs*, pages 445–457. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.
30. N. Kobayashi, K. Matsuda, and A. Shinohara. Functional programs as compressed data. In *Proceedings of PEPM 2012*, pages 121–130. ACM Press, 2012.
31. M. Lohrey. On the parallel complexity of tree automata. In *Proceedings of RTA 2001*, volume 2051 of *Lecture Notes in Computer Science*, pages 201–215. Springer, 2001.
32. M. Lohrey. Leaf languages and string compression. *Information and Computation*, 209(6):951–965, 2011.
33. M. Lohrey. *The Compressed Word Problem for Groups*. Springer, 2014.
34. M. Lohrey. Grammar-based tree compression. In *Proceedings of DLT 2015*, volume 9168 of *Lecture Notes in Computer Science*, pages 46–57. Springer, 2015.
35. M. Lohrey and S. Maneth. The complexity of tree automata and XPath on grammar-compressed trees. *Theoretical Computer Science*, 363(2):196–210, 2006.
36. M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using RePair. *Information Systems*, 38(8):1150–1167, 2013.
37. M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *Journal of Computer and System Sciences*, 78(5):1651–1669, 2012.
38. M. Lohrey and C. Mathissen. Isomorphism of regular trees and words. *Information and Computation*, 224:71–105, 2013.
39. J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, 2001.
40. G. Navarro, A. Ordóñez Pereira. Faster compressed suffix trees for repetitive text collections. In *Proceedings of SEA 2014*, volume 8504 of *Lecture Notes in Computer Science*, pages 424–435. Springer, 2014.
41. W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302(1–3):211–222, 2003.
42. H. Sakamoto. A fully linear-time approximation algorithm for grammar-based compression. *Journal of Discrete Algorithms*, 3(2-4):416–430, 2005.
43. M. Schmidt-Schauß. Linear compressed pattern matching for polynomial rewriting (extended abstract). In *Proceedings of TERMGRAPH 2013*, volume 110 of *EPTCS*, pages 29–40, 2013.
44. S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.
45. H. Vollmer. *Introduction to Circuit Complexity*. Springer, 1999.