Tree compression using string grammars

Moses Ganardi, Danny Hucke, Markus Lohrey, and Eric Noeth
University of Siegen, Germany
{ganardi,hucke,lohrey,eric.noeth}@eti.uni-siegen.de

Abstract. We study the compressed representation of a ranked tree by a (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 turn out to 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 and Horton-Strahler number 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.

1 Introduction

Grammar-based compression has emerged to 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 $A_0 \rightarrow ab$ and $A_i \rightarrow A_{i-1}A_{i-1}$ 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 $2^{O(n)}$. 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 analyzed. Prominent grammar-based string compressors are for instance LZ78, RePair, and BISECTION, see [8] for more details. The theoretically best known polynomial time grammar-based compressors [5,17,22] approximate the size of a smallest SLP up to a factor $O(\log(n/g))$, where $g$ is the size of a smallest SLP for the input string.

Motivated by applications where large tree structured data occur, like XML processing, grammar-based compression has been extended to trees [5,17,22]. In those papers, straight-line linear context-free tree grammars were used. These are grammars that produce only a single tree and are also known as tree straight-line programs (TSLPs). TSLPs generalize dags (directed acyclic graphs), which are widely used as a compact tree representation. Whereas dags only allow to share repeated subtrees, TSLPs can also share repeated internal tree patterns. The grammar-based tree compressor from [17] achieves an approximation ratio of $O(\log n)$ (for a fixed set of node labels) and the algorithm from [14] produces for every tree (again for a fixed set of node labels) a TSLP of size $O(\frac{\log n}{\log \log n})$, which is worst-case optimal.

The probably simplest approach to tree compression is to encode trees by strings and then use a string compressor to compact the tree. This is also natural in many applications, where trees are by default represented by strings (this is done for example with XML documents). There are several standard string encodings of trees. For trees over a ranked alphabet (which we deal with in this paper), preorder traversals or postorder traversals are widely used and quite natural. For instance, the preorder (resp., postorder) traversal of the tree $f(g(a), f(a, b))$ is the string $fgafab$ (resp., $agfabf$). Note that for trees over a ranked alphabet the initial tree can be obtained from its preorder (resp., postorder) traversal. In this paper, we only use preorder traversals for encoding

⋆ The third and fourth author are supported by the DFG-project LO 748/10-1 (QUANT-KOMP).
1 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.
trees, but by symmetry all results can be also stated for postorder traversals. In fact, we will identify a ranked tree with its preorder traversal string.

We compress a tree \( t \) by using a (string) SLP for the preorder traversal of \( t \). One can easily check in linear time, whether a given SLP produces the preorder traversal of a tree (Theorem \[1\]). It was shown in \[6\] that from a given TSLP \( \mathcal{A} \) of size \( m \) for a tree \( t \) one can efficiently construct an SLP of size \( \mathcal{O}(m \cdot r) \) for the traversal of \( t \), where \( r \) is the maximal rank occurring in \( t \) (i.e., the maximal number of children of a node). Hence, a smallest SLP for the traversal of \( t \) cannot be much larger than a smallest TSLP for \( t \). Our first main result shows that SLPs are in general 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 in \( \Omega(2^{n/2}) \). We also match this lower bound by an upper bound: Given an SLP \( \mathcal{A} \) 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 \( \mathcal{O}(m \cdot h \cdot r) \).

We also study algorithmic problems for trees that are encoded by SLPs. It turns out that several important algorithmic problems on trees can be solved in polynomial time if the input tree is represented by an SLP. These include the computation of the height and the Horton-Strahler number of the tree, the depth of a given node, navigation in the tree and simple evaluation problems like the evaluation of Boolean expressions that are represented by SLPs. On the other hand, there also exist problems that are easy (polynomial time solvable) for TSLP-represented trees but difficult for SLP-represented trees: Examples for such problems are the membership problem for tree automata, pattern matching and evaluation of max-plus expressions.

2 Preliminaries

Let \( \Sigma \) 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 \( w[\hat{i}] = w[1 : i] \) and \( w[i] = w[i : w] \). With \( \text{rev}(w) = a_n \cdots a_1 \) 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 \( 1 \leq i \leq \min\{|u|, |v|\}\).

Complexity Classes. We assume familiarity with the basic classes from complexity theory, in particular \( \mathsf{P} \), \( \mathsf{NP} \) and \( \mathsf{PSPACE} \). The counting class \( \#\mathsf{P} \) contains all functions \( f : \Sigma^* \rightarrow \mathbb{N} \) 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 \( \mathsf{PP} \) 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. Finally the levels of the counting hierarchy \( \mathsf{C}_i^{\mathsf{P}} \) \((i \geq 0)\) are inductively defined as follows: \( \mathsf{C}_0^{\mathsf{P}} = \mathsf{P} \) and \( \mathsf{C}_{i+1}^{\mathsf{P}} = \mathsf{PP}^{\mathsf{C}_i^{\mathsf{P}}} \) for all \( i \geq 0 \). It is known that \( \mathsf{C}_i^{\mathsf{P}} \subseteq \mathsf{PSPACE} \) for all \( i \geq 0 \). More details can be found in \[3\].

Trees. A ranked alphabet \( \mathcal{F} \) is a finite set of symbols where every symbol \( f \in \mathcal{F} \) has a rank \( \text{rank}(f) \in \mathbb{N} \). 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 \( ft_1 \cdots t_n \in \mathcal{T}(\mathcal{F}) \).

We call a string \( s \in \mathcal{F}^+ \) a fragment if there exists a tree \( t \in \mathcal{T}(\mathcal{F}) \) and a non-empty string \( x \in \mathcal{F}^+ \) such that \( sx = t \). Note that the empty string \( \varepsilon \) 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 \( \text{gaps}(s) \). The number of gaps of the empty string is defined as 0. The following lemma states that \( \text{gaps}(s) \) is indeed well-defined.

Lemma 1. The following statements hold:
Example 1. Let \( f \) be the tree depicted in Figure 1 with \( f \in F_2 \) and \( a \in F_0 \). Its height is 4. All prefixes (including the empty word, excluding the full word) of \( f \) are fragments. The fragment \( s = f a a f f a f f a a \) is also depicted in Figure 1 in a graphical way. The dashed edges visualize the gaps. We have gaps(s) = 4. For the factor \( u = a a f f f a \) of \( t \) we have \( c(u) = (2, 3) \). The children of node 5 (the third \( f \)-labelled node) are 6 and 11.

**Fig. 1.** The tree \( t \) from Example 1 and the tree fragment corresponding to the fragment \( f f a a f f a f f a a \).

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

Since \( T(F) \) is prefix-free we immediately get:

**Lemma 2.** For every \( w \in F^* \) there exist unique \( n \geq 0 \), \( t_1, \ldots, t_n \in T(F) \) and a unique fragment \( s \) such that \( w = t_1 \cdots t_n s \).

Let \( w \in F^* \) and let \( w = t_1 \cdots t_n s \) as in Lemma 2. We define \( c(w) = (n, \text{gaps}(s)) \). The number \( n \) counts the number of full trees in \( w \) and \( \text{gaps}(s) \) is the number of trees missing to make the fragment \( s \) a tree, too.

For better readability, we occasionally write a tree \( ft_1 \cdots t_n \) with \( f \in F_n \) and \( t_1, \ldots, t_n \in T(F) \) 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 where the set of nodes of a tree \( t \) is the set of positions \( \{1, \ldots, |t|\} \) of the string \( t \). The root node is 1. If \( t \) factorizes as \( u t_1 \cdots t_n v \) for \( u, v \in F^* \), \( f \in F_n \), and \( t_1, \ldots, t_n \in T(F) \), then the \( n \) children of node \( |u| + 1 \) are \( |u| + 2 + \sum_{i=1}^{k} |t_i| \) for \( 0 \leq k \leq n - 1 \).

We define the depth of a node in \( t \) (number of edges from the root to the node) and the height of \( t \) (maximal depth of a node) as usual. Note that the tree \( t \) as a string is simply the preorder traversal of the tree \( t \) seen in its standard graph-theoretic interpretation.

**Example 1.** Let \( t = f f a a f f f a a a = f(f(a, a), f(f(f(a, a), a), a)) \) be the tree depicted in Figure 1 with \( f \in F_2 \) and \( a \in F_0 \). Its height is 4. All prefixes (including the empty word, excluding the full word) of \( t \) are fragments. The fragment \( s = f f a a f f a f f a a \) is also depicted in Figure 1 in a graphical way. The dashed edges visualize the gaps. We have gaps(s) = 4. For the factor \( u = a a f f f a \) of \( t \) we have \( c(u) = (2, 3) \). The children of node 5 (the third \( f \)-labelled node) are 6 and 11.

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

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

Every nonterminal \( A \in N \) produces a unique string \( \text{val}_A(A) \in \Sigma^* \). The string defined by \( A \) is \( \text{val}(A) = \text{val}_A(S) \). We usually omit the subscript \( A \) when the context is clear. The size of the SLP \( A \) is \( |A| = \sum_{(A \rightarrow w) \in P} |w| \). One can transform an SLP \( A = (N, \Sigma, P, S) \) which produces a nonempty word in linear time into *Chomsky normal form*, i.e. for each production \( (A \rightarrow w) \in P \), either \( w \in \Sigma \) or \( w = BC, B, C \in N \).

For an SLP \( A \) of size \( n \) we have \( |\text{val}(A)| \leq 2^\Theta(n) \), and there exists a family of SLPs \( A_n \) (\( n \geq 1 \)) such that \( |A_n| \in \Theta(n) \) and \( |\text{val}(A)| = 2^n \). Hence, SLPs allow exponential compression.

The following lemma summarizes known results about SLPs which we will use throughout the paper, see e.g. [20].
Lemma 3. There are polynomial time algorithms for the following problems:

1. Given an SLP $\mathcal{A}$, compute the set of symbols occurring in $\text{val}(\mathcal{A})$.
2. Given an SLP $\mathcal{A}$ with terminal alphabet $\Sigma$ and a subset $\Gamma \subseteq \Sigma$, compute the number of occurrences of symbols from $\Gamma$ in $\text{val}(\mathcal{A})$.
3. Given an SLP $\mathcal{A}$ with terminal alphabet $\Sigma$, a subset $\Gamma \subseteq \Sigma$, and a number $i$, compute the position of the $i^{th}$ occurrence of a symbol from $\Gamma$ in $\text{val}(\mathcal{A})$ (if it exists).
4. Given an SLP $\mathcal{A}$ and $i, j \in \{1, \ldots, |\text{val}(\mathcal{A})|\}$ where $i \leq j$, compute an SLP for $\text{val}(\mathcal{A})[i : j]$.

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. Let further $\mathcal{X} = \{x_1, x_2, \ldots\}$ 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 $\mathcal{A}$, or short TSLP, is a tuple $\mathcal{A} = (\mathcal{V}, \mathcal{F}, P, S)$, 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 of the form $A(x_1, \ldots, x_n) \rightarrow t$ (which is also briefly written as $A \rightarrow t$), where $n \geq 0$, $A \in \mathcal{V}_n$ and $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{V} \cup \{x_1, \ldots, x_n\})$ is a tree in which every parameter $x_i$ ($1 \leq i \leq n$) occurs at most once, such that:

- For every $A \in \mathcal{V}_n$ there exists exactly one production of the form $A(x_1, \ldots, x_n) \rightarrow t$, and
- the binary relation $\{(A, B) \in \mathcal{V} \times \mathcal{V} \mid (A \rightarrow t) \in P, B$ is a label in $t\}$ is acyclic.

These conditions ensure that exactly one tree $\text{val}_\mathcal{A}(A) \in \mathcal{T}(\mathcal{F} \cup \{x_1, \ldots, x_n\})$ is derived from every nonterminal $A \in \mathcal{V}_n$ by using the rules as rewriting rules in the usual sense. As for SLPs, we omit the subscript $\mathcal{A}$ when the context is clear. The tree defined by $\mathcal{A}$ is $\text{val}_\mathcal{A}(A) = \text{val}_\mathcal{A}(S)$. The size $|\mathcal{A}|$ of a TSLP $\mathcal{A} = (\mathcal{V}, \mathcal{F}, P, S)$ is $|\mathcal{A}| = \sum_{(A \rightarrow t) \in P} |t|$. We call a TSLP monadic if every nonterminal has rank at most one. One can transform every TSLP into a monadic one of size $\mathcal{O}(|\mathcal{A}| \cdot r)$, where $r$ is the maximal rank of a terminal in $\mathcal{A}$ [23]. TSLPs, where every nonterminal has rank 0 correspond to dags (the nodes of the dag are the nonterminals of the TSLP).

For a TSLP $\mathcal{A}$ of size $n$ we have $|\text{val}_\mathcal{A}(A)| \in 2^\mathcal{O}(n)$, and there exists a family of TSLPs $\mathcal{A}_n$ ($n \geq 1$) such that $|\mathcal{A}_n| \in \mathcal{O}(n)$ and $|\text{val}_\mathcal{A}(A)| = 2^n$. 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. [21].

For every word $w$ (resp., tree $t$) there exists a smallest SLP (resp., TSLP) $\mathcal{A}$. It is known that, unless $\mathcal{P} = \mathcal{NP}$, there is no polynomial time algorithm that finds a smallest SLP (resp., TSLP) for a given word [8] (resp. tree).

3 Checking whether an SLP produces a tree

In this section we show that, given an SLP $\mathcal{A}$ and a ranked alphabet $\mathcal{F}$, we can verify in time linear in $|\mathcal{A}|$, whether $\text{val}(\mathcal{A}) \in \mathcal{T}(\mathcal{F})$. 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 $S \rightarrow fS^n$ 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 [719].

Theorem 1. Given an SLP $\mathcal{A}$, one can check in time linear in the size of $\mathcal{A}$ whether $\text{val}(\mathcal{A}) \in \mathcal{T}(\mathcal{F})$.

Proof. Let $\mathcal{A} = (N, \mathcal{F}, P, S)$ be in Chomsky normal form and let $A \in N$. Due to Lemma [2] we know that $\text{val}(A)$ is the concatenation of trees and a (possibly empty) fragment. Define $c(A) := c(\text{val}(A))$. 
Then \( \text{val}(A) \in T(F) \) if and only if \( c(S) = (1, 0) \). Hence, it suffices to compute \( c(A) \) for all nonterminals \( A \in N \). We do this bottom-up. If \( (A \rightarrow f) \in P \) with \( f \in F_n \), then we have

\[
c(A) = \begin{cases} 
(1, 0) & \text{if } n = 0 \\
(0, n) & \text{otherwise.}
\end{cases}
\]

Now consider a nonterminal \( A \) with the rule \( (A \rightarrow BC) \in P \), and let \( c(B) = (b_1, b_2) \), \( c(C) = (c_1, c_2) \). We claim that

\[
c(A) = \begin{cases} 
(b_1 + c_1 - \max\{1, b_2\} + 1, c_2) & \text{if } b_2 \leq c_1 \\
(b_1, c_2 + b_2 - c_1 - \min\{1, c_2\}) & \text{otherwise.}
\end{cases}
\]

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

Case \( b_2 \leq c_1 \): If \( b_2 \geq 1 \), then the string \( st_1' \cdots t_{b_2}' \) is a tree, and thus \( \text{val}(A) \) 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 \( \text{val}(A) \) 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 \( c_2 = 0 \). \( \Box \)

### 4 SLPs versus TSLPs

In [6] it is shown that a TSLP \( A \) producing a tree \( t \in T(F) \) can always be transformed into an SLP of size \( O(|A| \cdot r) \) producing \( t \), where \( r \) is the maximal rank of a label occurring in \( t \). 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 \( O(\log n) \). It is less obvious that such an exponential gap can be also 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 \( A \) for \( t \) into a TSLP of size \( O(|A| \cdot h \cdot r) \) for \( t \), where \( h \) is the height of \( t \) and \( r \) is the maximal rank of a label occurring in \( t \).

#### 4.1 Worst-case comparison of SLPs and TSLPs

For two words \( u, v \in \{0, 1\}^* \) of the same length, we denote by \( u \land v \) the result of the bitwise AND. We use the following result from [4] for the previously mentioned worst-case construction of a family of binary trees:

**Theorem 2 (Thm. 2 from [4]).** 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 \( n^{o(1)} \), but the smallest SLP for \( u_n \land v_n \) has size \( \Omega(2^{n/2}) \).

For two given words \( u = i_1 \cdots i_n, v = j_1 \cdots j_n \in \{0, 1\}^* \) we define the comb tree

\[ t(u, v) = f_{i_1}(f_{i_2}(\ldots f_{i_n}(\$, j_n) \ldots j_2), j_1) \]

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[2] for an illustration.

**Theorem 3.** For every \( n > 0 \) there exists a tree \( t_n \) such that the size of the smallest SLP for \( t_n \) is polynomial in \( n \), but the size of the smallest TSLP for \( t_n \) is in \( \Omega(2^{n/2}) \).
Without loss of generality we assume that $A$ is in Chomsky normal form. For every nonterminal $A$ of $\mathcal{A}$ with $c(A) = (a_1, a_2)$ we introduce $a_1$ nonterminals $A_1, \ldots, A_n$ 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 $A \rightarrow f$ with $f \in \mathcal{F}_n$ we add to $\mathcal{B}$ the TSLP-rule $A_1 \rightarrow f$ if $n = 0$ or $A'(x_1, \ldots, x_n) \rightarrow f(x_1, \ldots, x_n)$ if $n \geq 1$. Now consider a rule of the form $A \rightarrow BC$ with $c(B) = (b_1, b_2)$ and $c(C) = (c_1, c_2)$. 

\hspace{1cm}

\begin{center}
\textbf{Fig. 2.} The comb tree $t(u, v)$ for $u = i_1 \cdots i_n$ and $v = j_1 \cdots j_n$
\end{center}

\textbf{Proof.} Let us fix an $n$ and let $u_n$ and $v_n$ be the aforementioned strings from Theorem 2. Let $|u_n| = |v_n| = m$. Consider the comb tree $t_n := t(u_n, v_n)$. Note that $t_n = f_1 \cdots f_m \lambda \rev(v_n)$, where $u_n = i_1 \cdots i_m$. By Theorem 2 there exist SLPs of size $n^{O(1)}$ for $u_n$ and $v_n$, and these SLPs easily yield an SLP of size $n^{O(1)}$ for $t_n$.

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

Let $\mathcal{A}$ be a TSLP for $t_n$. By 23 we can transform $\mathcal{A}$ into a monadic TSLP $\mathcal{A}'$ for $t_n$ of size $O(|\mathcal{A}|)$. We transform the TSLP $\mathcal{A}'$ into an SLP of the same size for $u_n \land v_n$. 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 $A \rightarrow j$ ($j \in \{0, 1\}$) and replace the occurrences of $A$ by $j$ in all right-hand sides. 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 subtree $\text{val}(A)$. 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 $g_1(g_2(\ldots (g_k(x), j_2), j_1) \cdot \cdots \cdot j_n) \cdot \cdots \cdot 1)$ ($g_i \in \{f_0, f_1\}$ 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). Therefore we can obtain an SLP for $u_n \land v_n$ by replacing every production $A(x) \rightarrow t(x)$ by $A \rightarrow \lambda(t(x))$, 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 $\lambda(f_i(s(x), j)) = (i \wedge j)\lambda(s(x))$. The production for $S$ must be of the form $S \rightarrow t(\$)$ for a term $t(x)$ and we replace it by $S \rightarrow \lambda(t(x))\$.

Note that the height of the tree $t_n$ in Theorem 3 is linear in the size of $t_n$. By the result from the next section, large height and rank are always responsible for the exponential succinctness gap between SLPs and TSLPs.

\subsection{Conversion of SLPs to TSLPs}

\textbf{Theorem 4.} Let $t \in \mathcal{T}(\mathcal{F})$ be a tree of height $h$ and maximal rank $r$, and let $\mathcal{A}$ be an SLP for $t$ with $|\mathcal{A}| = m$. Then there exists a TSLP $\mathcal{B}$ with $\text{val}(\mathcal{B}) = t$ such that $|\mathcal{B}| \in O(m \cdot h \cdot r)$, which can be constructed in time $O(m \cdot h \cdot r)$.

\textbf{Proof.} Without loss of generality we assume that $A$ is in Chomsky normal form. For every nonterminal $A$ of $\mathcal{A}$ with $c(A) = (a_1, a_2)$ we introduce $a_1$ nonterminals $A_1, \ldots, A_n$ 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 $A \rightarrow f$ with $f \in \mathcal{F}_n$ we add to $\mathcal{B}$ the TSLP-rule $A_1 \rightarrow f$ if $n = 0$ or $A'(x_1, \ldots, x_n) \rightarrow f(x_1, \ldots, x_n)$ if $n \geq 1$. Now consider a rule of the form $A \rightarrow BC$ with $c(B) = (b_1, b_2)$ and $c(C) = (c_1, c_2)$. 

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

$$
A_i \rightarrow B_i \quad \text{for } 1 \leq i \leq b_1 \\
A_{b_1+i} \rightarrow C_i \quad \text{for } 1 \leq i \leq c_1 \\
A'(x_1, \ldots, x_{c_2}) \rightarrow C'(x_1, \ldots, x_{c_2}) \quad \text{if } c_2 > 0
$$

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

$$
A_i \rightarrow B_i \quad \text{for } 1 \leq i \leq b_1 \\
A_{b_1+i} \rightarrow B'(C_1, \ldots, C_{b_2}) \\
A_{b_1+b_2+i} \rightarrow C_{b_2+i} \quad \text{for } 1 \leq i \leq c_1 - b_2 \\
A'(x_1, \ldots, x_{c_2}) \rightarrow C'(x_1, \ldots, x_{c_2}) \quad \text{if } c_2 > 0
$$

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

$$
A_i \rightarrow B_i \quad \text{for } 1 \leq i \leq b_1 \\
A'(x_1, \ldots, x_d) \rightarrow B'(C_1, \ldots, C_{c_1}, x_1, \ldots, x_d) \quad \text{if } c_2 = 0 \\
A'(x_1, \ldots, x_{c_2+d-1}) \rightarrow B'(C_1, \ldots, C_{c_1}, C'(x_1, \ldots, x_{c_2}), x_{c_2+1}, \ldots, x_{c_2+d-1}) \quad \text{if } c_2 > 0
$$

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 $t_i \in \mathcal{T}(F)$ 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 $n \leq h \cdot r$.

\[\square\]

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 Tree Navigation

We start with the problem of navigating in a tree that is represented by an SLP. Recall that we identify the nodes of a tree $t$ with the positions $1, \ldots, |t|$ in the string $t$.

Theorem 5. Given an SLP $\mathcal{A}$ for a tree $t \in \mathcal{T}(F)$ and $i, k \in \mathbb{N}$ (given in binary encoding) where $1 \leq i \leq |t|$, the following problems are solvable in polynomial time:

(a) Compute the parent node of node $i > 1$ in $t$.
(b) Compute the $k^{th}$ child of node $i$ in $t$, if it exists.
(c) Compute the $k^{th}$ next (resp., previous) sibling of node $i$ in $t$, if it exists.

Proof. Let $t = \text{val}(\mathcal{A})$. For point (i) we perform a binary search for $j \in \{1, \ldots, i-1\}$ such that $c(t[j : i-1]) = (0, n)$ and $c(t[j+1 : i-1]) = (m, 0)$ holds for some $n, m \in \mathbb{N}$. The first condition ensures that $j$ is an ancestor of $i$ and the second condition ensures that it is the parent node. The search is done as follows. We first calculate $c(t[j : i-1]) = (k_1, k_2)$, where $j = \lfloor \frac{t}{2} \rfloor$. If $k_1 > 0$ and $k_2 = 0$ we continue our search to the left. If $k_1 > 0$ and $k_2 > 0$ we continue our search to the right. If $k_1 = 0$ and $k_2 > 0$ we calculate $c(t[j+1 : i-1])$. If this is of the form $(m, 0)$ we have found the parent node, otherwise we continue our search to the right.
The proof for (b) is similar to (a). Here we perform a binary search for \( j \in \{i + 1, \ldots, |t|\} \) and compute in each step \( c(t[i + 1 : j]) = (k_1, k_2) \). If \( k_1 \) is smaller (greater) than \( k - 1 \), we move to the right (left). In case \( k_1 = k - 1 \), if \( k_2 = 0 \) then \( j + 1 \) is the \( k^{th} \) child, otherwise we move to the left.

For (c) we compute the parent node \( j \) of node \( i \) using (a) and compute \( c(t[j+1 : i-1]) = (n, m) \). Then, \( i \) is the \((n + 1)^{st}\) child of its parent node (note that we must have \( m = 0 \)). Using (b) we can navigate from node \( j \) to its \((n + 1 + k)^{th}\) child. Navigation to previous siblings can be done similarly.

\[ \square \]

5.2 Pattern matching

In contrast to navigation problems, simple pattern matching problems become quite difficult 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 \( A \) producing a tree \( t \in \mathcal{T}(\mathcal{F}) \), is there a substitution \( \sigma : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}) \) 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 \[27\] (even if the pattern tree \( s \) is given by a TSLP as well).

**Theorem 6.** The pattern matching problem for SLP-compressed trees is NP-complete. 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 5.

By \[29\] Theorem 3.13 it is NP-complete to decide for given SLPS \( A, B \) over \( \{0,1\} \) with \(|\text{val}(A)| = |\text{val}(B)|\) whether there exists a position \( i \) such that \( \text{val}(A)[i] = \text{val}(B)[i] = 1 \). This question can be reduced to the pattern matching problem with a fixed pattern. One can compute in polynomial time from \( A \) and \( B \) an SLP \( T \) for the comb tree \( t(\text{val}(A), \text{val}(B)) \). There exists a position \( i \) such that \( \text{val}(A)[i] = \text{val}(B)[i] = 1 \) if and only if the pattern \( f_1(x, 1) \) occurs in \( t(\text{val}(A), \text{val}(B)) \). \[ \square \]

5.3 Tree evaluation problems

In the following we present a general framework to define and solve evaluation problems on SLP-compressed trees. We assign to each alphabet symbol of rank \( n \) an \( n \)-ary operator which defines the value of a tree by evaluating it bottom-up. This approach includes natural tree problems like computing the height of a tree, evaluating a Boolean expression or determining whether a fixed tree automaton accepts a given tree. Note that we only work with operators on integers but other domains with an appropriate encoding of the elements are also possible. To be able to consider arbitrary arithmetical 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 1.** 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 \( f^\mathcal{I} : \mathcal{D}^n \rightarrow \mathcal{D} \) 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 2.** The \( \mathcal{I} \)-evaluation problem for SLP-compressed trees is the following problem: Given an SLP \( A \) over \( \mathcal{F} \) with \( \text{val}(A) \in \mathcal{T}(\mathcal{F}) \), compute \( \text{val}(A)^\mathcal{I} \).
Reduction to caterpillar trees. In this section, we reduce the 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 $s^2 \in \mathcal{F}^*$ denotes the unique string obtained from $s$ by replacing every maximal substring $t \in \mathcal{T}(\mathcal{F})$ of $s$ by its value $t^2$. 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 $s^2 = m_1 \cdots m_n u^2$ with $m_1, \ldots, m_n \in \mathcal{D}$. Since $u$ is a fragment, the string $u^2$ is the fragment of a caterpillar tree (briefly, caterpillar fragment in the following).

Our reduction to caterpillar trees only works for interpretations that satisfy a certain growth condition. We say that an interpretation $I$ is polynomially bounded, if there exist constants $\alpha, \beta \geq 0$ such that for every tree $t \in \mathcal{T}(\mathcal{F})$,

\[ |t^2| \leq \left( \beta \cdot |t| + \sum_{i \in L} |t[i]| \right)^\alpha \]

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 $\text{val}(\mathbb{A}) \in \mathcal{T}(\mathcal{F})$, the length of the binary encoding of $\text{val}(\mathbb{A})^2$ is polynomially bounded in $|\mathbb{A}|$ and the binary lengths of the constants that appear in $\mathbb{A}$.

**Theorem 7.** Let $I$ be a polynomially bounded interpretation. Then the I-evaluation problem for SLP-compressed trees is polynomial time Turing-reducible to the 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 $\mathbb{A} = (N, \Sigma, P, S)$ is an SLP where $P$ may also contain rules of the form $A \rightarrow B[i : j]$ where $A, B \in N$ and $i, j \geq 0$. Here we let $\text{val}(A) = \text{val}(B)[i : j]$. It is known \cite{12} (see also \cite{20}) that a given composition system can be transformed in polynomial time into an SLP with the same value. One can also allow mixed rules $A \rightarrow X_1 \cdots X_n$ 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 $\mathbb{A} = (N, \mathcal{F}, P, S)$ be the input SLP in Chomsky normal form. We use the notation $c(A) = c(\text{val}(\mathbb{A}))$ as in the proof of Theorem 1. We will compute a composition system where for each nonterminal $A \in N$ there are nonterminals $A_1$ and $A_2$ in the composition system such that the following holds: Assume that $\text{val}(A) = t_1 \cdots t_n s$, where $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ and $s$ is a fragment. Hence, $c(A) = (n, \text{gaps}(s))$. Then we will have

- $\text{val}(A_1) = t_1^2 \cdots t_n^2 \in \mathcal{D}^*$, and
- $\text{val}(A_2) = s^2$.

In particular, $\text{val}(A_1) \text{val}(A_2) = \text{val}(A)^2$ and $\text{val}(\mathbb{A})^2$ is given by the single number in $\text{val}(S_1)$.

The computation is straightforward for rules of the form $A \rightarrow f$ with $A \in N$ and $f \in \mathcal{F}$. If $\text{rank}(f) = 0$, then $\text{val}(A_1) = f$ and $\text{val}(A_2) = \varepsilon$. If $\text{rank}(f) > 0$, then $\text{val}(A_1) = \varepsilon$ and $\text{val}(A_2) = f$.

For a nonterminal $A \in N$ with the rule $A \rightarrow BC$ we do a case distinction depending on $c(B) = (b_1, b_2)$ and $c(C) = (c_1, c_2)$.

**Case $b_2 \leq c_1$:** Then concatenating $\text{val}(B)$ and $\text{val}(C)$ yields a new tree $t_{\text{new}}$ (or $\varepsilon$ in case $b_2 = 0$) in $\text{val}(A)$. Notice that $t_{\text{new}}^2$ is the value of the tree $\text{val}(B_2) \text{val}(C_1)[1 : b_2]$. Hence we can compute $t_{\text{new}}^2$ in polynomial time by computing an SLP that produces $\text{val}(B_2) \text{val}(C_1)[1 : b_2]$ and querying the oracle for the caterpillar tree. We add the following rules to the composition system:

\[
A_1 \rightarrow B_1 \ t_{\text{new}}^2 C_1[b_2 + 1 : c_1] \\
A_2 \rightarrow C_2
\]

**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., $\text{val}(C_1) = \varepsilon$), then we add the productions $A_1 \rightarrow B_1$ and $A_2 \rightarrow B_2C_2$. Now assume that $c_1 > 0$. Consider the fragment

\[ s = \text{val}(B_2) \text{val}(C_1) \text{val}(C_2). \]
Intuitively, this fragment $s$ is obtained by taking the caterpillar fragment $\text{val}(B_2)$, where the first $c_1$ many gaps are replaced by the constants from the sequence $\text{val}(C_1)$ and the $(c_1 + 1)^{\text{th}}$ gap is replaced by the caterpillar fragment $\text{val}(C_2)$, see Figure 3. If $s$ is not already a caterpillar fragment, then we have to replace the (unique) largest factor of $s$ which belongs to $T(F)$ by its value under $I$ to get $s'$. To do so we proceed as follows: Consider the tree $t' = \text{val}(B_2) \text{val}(C_1) \circ^{b_2-c_1}$, where $\circ$ is an arbitrary symbol of rank 0, and let $r = |\text{val}(B_2)| + c_1 + 1$ (the position of the first $\circ$ in $t'$). Let $q$ be the parent node of $r$, which can be computed in polynomial time by Theorem 5. 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 $A_1 \rightarrow B_1$ and $A_2 \rightarrow B_2C_1C_2$ to the composition system. Otherwise $p$ is the first symbol of the factor described above. Note that the position $p$ must belong to $\text{val}(B_2)$ and that $p$ has a next sibling $p'$, which is also computable by Theorem 5. The string $t_{\text{cat}} = (\text{val}(B_2) \text{val}(C_1))[p : p' - 1]$ is a caterpillar tree for which we can compute an SLP in polynomial time by $[12]$. Hence, using the oracle we can compute the value $t^I_{\text{cat}}$. By substituting $t_{\text{cat}}$ by $t^I_{\text{cat}}$, we obtain the partially evaluated caterpillar fragment corresponding to $A$. We then add the following rules to the composition system:

$$A_1 \rightarrow B_1$$
$$A_2 \rightarrow B_2[p : p - 1] t^I_{\text{cat}} B_2[p' : :] C_2.$$  

This completes the proof.  

**Polynomial time solvable evaluation problems.** We will now introduce a particular class of interpretations that allows to evaluate SLP-compressed caterpillar trees in polynomial time. Important evaluation problems like computation of the height or the depth of a node will be covered by our class.

**Theorem 8.** Let $I$ be an interpretation of $\mathcal{F}$ over $D \subseteq \mathbb{Z}$ such that for each $f \in \mathcal{F}_n$ with $n > 0$, there exists a polynomial time computable function $g : D \rightarrow D$ such that

- $f^I(d_1, \ldots, d_n) = g(\max\{d_1, \ldots, d_n\})$ for all $d_1, \ldots, d_n \in D$ and
- there exists an $\alpha \geq 0$ such that $d \leq g(d) \leq \alpha + d$ for all $d \in D$.

Then the $I$-evaluation problem for SLP-compressed trees can be solved in polynomial time.

**Proof.** Clearly, $I$ is polynomially bounded. By Theorem 7 it is enough to show how to evaluate a caterpillar tree $t$ given by an SLP $\mathcal{A}$ in polynomial time. Let $1 = u_1 < u_2 < \cdots < u_N$ be the nodes on the unique path in $t$ labelled with symbols of rank $> 0$ and for all $1 \leq i \leq N$ let $t_i$ be the subtree of $t$ rooted at $u_i$. We can assume that $t$ is not a single constant, i.e., $N \geq 1$. Note that
interprets constant symbols \( a \) at \( t \) occurs exactly once). Let \( d \) be the subtree of \( t \) equals to \( a \) in polynomial time if we know for each \( d \in \mathcal{D}_t \) the deepest node \( u_i \), which has a child labelled by \( d \). Let \( k \) be the maximal position in \( t \) where a symbol of rank \( > 0 \) occurs. The number \( k \) is computable in polynomial time by Lemma 3 (point 2 and 3). Note that \( t[1:k] \) is the fragment where all leaves right of the path \( u_1, \ldots, u_N \) are removed and \( t[k+1:] \in \mathcal{D}^* \) is the list of the removed leaf values (starting with the children of \( u_N \)). Again using Lemma 3 we compute for each \( d \in \mathcal{D}_t \) the position of its last occurrence in \( t[1:k] \) and its first occurrence in \( t[k+1:] \). Then using Theorem 5 we compute the parent nodes of those two nodes in \( \text{val}(\mathcal{A}) \) and take the maximum (i.e., the deeper one) of both.

**Corollary 1.** The height of a tree given by an SLP \( \mathcal{A} \) is computable in polynomial time.

**Proof.** The height of a tree is given by the value under the interpretation \( \mathcal{I} \) with \( a^\mathcal{I} = 0 \) for \( a \in \mathcal{F}_0 \) and \( f^\mathcal{I}(a_1,\ldots,a_n) = 1 + \max\{a_1,\ldots,a_n\} \) for symbols \( f \in \mathcal{F}_n \) with \( n > 0 \).

**Corollary 2.** Given an SLP \( \mathcal{A} \) for a tree \( t \) and a node \( 1 \leq i \leq |t| \), one can compute the depth of \( i \) in polynomial time. Given an additional number \( k \), one can also compute in polynomial time the \( k^{th} \) ancestor of \( i \) in \( t \), if it exists.

**Proof.** First we replace the symbol at position \( i \) in \( t \) by a fresh symbol \( \diamond \) of the same rank. An SLP \( \mathcal{B} \) for the resulting tree can be computed in polynomial time from \( \mathcal{A} \). We define an interpretation \( \mathcal{I} \) over the domain \( \mathbb{N} \cup \{-1\} \) as follows: We interpret \( \diamond \) by the constant 0-function and all other symbols \( f \in \mathcal{F}_n \) by

\[
f^\mathcal{I}(a_1,\ldots,a_n) = \begin{cases} \ -1 & \text{if } n = 0 \text{ or } a_1 = \cdots = a_n = -1, \\ 1 + \max\{a_1,\ldots,a_n\} & \text{otherwise.} \end{cases}
\]

The depth of \( i \) is \( \text{val}(\mathcal{B})^\mathcal{I} \).

For the computation of the \( k^{th} \) ancestor we consider the tree \( s = t[1:i-1] \circ^m \) where \( m = \text{gaps}(t[1:i-1]) \). Similar to Theorem 5 we perform a binary search for \( j \in \{1,\ldots,i-1\} \) and compute in each step the depth \( d \) of node \( i \) in the tree \( s \). If \( d \) is smaller (greater) than \( k \), we move to the left (right). Otherwise, if \( d = k \) we compute \( c(s[j:i-1]) = (k_1, k_2) \). If \( k_1 = 0 \) then \( j \) is the \( k^{th} \) ancestor of \( i \), otherwise we continue the binary search to the right.

An interesting parameter of a tree \( t \) is its Horton-Strahler number or Strahler number, see [10] 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 [10], e.g., to calculate the minimum number of registers required to evaluate an arithmetical expression [11].

**Theorem 9.** Given an SLP \( \mathcal{A} \) 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 those used in Theorem 8. 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 8 must be slightly modified by considering for each \( d \in \mathcal{D} \) 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. 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 the same as in the proof of Theorem 8. □
One can extend Theorem 8 to a larger class of interpretations covering the Strahler number by adding another input variable to the functions \( g \) in Theorem 8 which is 1 if the maximum value among the children occurs more than once, and 0 otherwise. Since the proof is very technical, we omit it here.

If the interpretation \( I \) is clear from the context, we also speak of the problem of \textit{evaluating SLP-compressed \( F \)-trees}. In the following theorem the interpretation is given by the Boolean operations \( \land \) and \( \lor \) over \( \{0, 1\} \).

**Theorem 10.** \textit{Evaluating SLP-compressed \( \{\land, \lor, 0, 1\} \)-trees can be done in polynomial time.}

**Proof.** Let \( \mathcal{A} \) be an SLP over \( \{\land, \lor, 0, 1\} \) such that \( \text{val}(\mathcal{A}) \) is a caterpillar tree. Define a \textit{left caterpillar tree} to be a tree of the form \( uv \), where \( u \in \{\land, \lor\}^* \), \( v \in \{0, 1\}^* \) and \( |v| = |u| + 1 \). That means that the main branch of the caterpillar tree grows to the left. The evaluation of \( \text{val}(\mathcal{A}) \) is done in two steps. In a first step, we compute in polynomial time from \( \mathcal{A} \) a new SLP \( \mathcal{B} \) such that \( \mathcal{B} \) is a left caterpillar tree and \( \text{val}(\mathcal{A})^2 = \text{val}(\mathcal{B})^2 \). In a second step, we show how to evaluate a left caterpillar tree.

\textbf{Step 1.} Since \( \text{val}(\mathcal{A}) \) is caterpillar tree, we have \( \text{val}(\mathcal{A}) = uv \) with \( u \in \{\land, \lor, \land0, \land1, \lor0, \lor1\}^* \cdot \{\land, \lor\}, \ v \in \{0, 1\}^* \) and \( |v| \) is 1 plus the number of occurrences of the symbols \( \land, \lor \) in \( u \) that are not followed by 0 or 1 in \( u \). We can compute bottom-up the length of the maximal suffix of \( \text{val}(\mathcal{A}) \) from \( \{0, 1\}^* \) in polynomial time. Hence, by Lemma \( 5 \) we can compute in polynomial time SLPs \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \) such that \( \text{val}(\mathcal{A}_1) = u \) and \( \text{val}(\mathcal{A}_2) = v \).

We will show how to eliminate all occurrences of the patterns \( \land0, \land1, \lor0, \lor1 \). For this, it is technically easier to replace every occurrence of \( \text{val}(\mathcal{A}) \) with \( \text{val}(\mathcal{B}) \), where \( o \in \{\land, \lor\} \) and \( a \in \{0, 1\} \). Let \( \varphi : \{\land, \lor, \land0, \land1, \lor0, \lor1\}^* \rightarrow \{\land, \lor, \land0, \land1, \lor0, \lor1\}^* \) be the mapping that replaces every occurrence of \( o \) by a new symbol \( c_o \) \( o \in \{\land, \lor\}, a \in \{0, 1\} \). This mapping can be realized by a deterministic rational transducer. Hence, using [4, Theorem 1], we can compute an SLP \( \mathcal{B}_1 \) for \( \varphi(\text{val}(\mathcal{A}_1)) \). We now compute, using Lemma \( 5 \) the position \( i \) in \( \text{val}(\mathcal{B}_1) \) of the first occurrence of a symbol from \( \{\land0, \lor1\} \). Next, we compute an SLP \( \mathcal{C}_1 \) for the prefix \( \text{val}(\mathcal{B}_1)[i : i-1] \), i.e., we cut off the suffix starting in position \( i \). Moreover, we compute the number \( j \) of occurrences of symbols from \( \{\land, \lor\} \) in the suffix \( \text{val}(\mathcal{B}_1)[i:] \) and compute an SLP \( \mathcal{B}_2 \) for the string \( 0 \text{val}(\mathcal{A}_2)[j+2:] \) in case \( \text{val}(\mathcal{B}_1)[i] = \land0 \) and 1 \( \text{val}(\mathcal{A}_2)[j+2:] \) in case \( \text{val}(\mathcal{B}_1)[i] = \lor1 \). Then \( \text{val}(\mathcal{A}) \) evaluates to the same truth value as \( \varphi^{-1}(\text{val}(\mathcal{C}_1)) \text{val}(\mathcal{B}_2) \). The reason for this is that \( \varphi^{-1}(\text{val}(\mathcal{B}_1)[i:]) \text{val}(\mathcal{A}_2)[j+1] \) is a tree which evaluates to 0 (resp., 1) if \( \text{val}(\mathcal{B}_1)[i] = \land0 \) (resp., \( \text{val}(\mathcal{B}_1)[i] = \lor1 \)), because \( 0 \land x = 0 \) (resp., \( 1 \lor x = 1 \)).

Note that \( \varphi^{-1}(\text{val}(\mathcal{C}_1)) \text{val}(\mathcal{B}_2) \) is a caterpillar tree, where \( \text{val}(\mathcal{C}_1) \in \{\land, \lor, \land0, \lor0\}^* \) and \( \text{val}(\mathcal{B}_2) \in \{0, 1\}^* \). Since \( 1 \land x = x \) (resp., \( 0 \lor x = x \)), we can delete in the string \( \text{val}(\mathcal{C}_1) \) all occurrences of the symbols \( \land \) and \( \lor0 \) without changing the final truth value. Let \( \mathcal{D}_1 \) be an SLP for the resulting string, which is easy to compute from \( \mathcal{C}_1 \). Then \( \text{val}(\mathcal{D}_1) \text{val}(\mathcal{B}_2) \) is indeed a left caterpillar tree.

See Figure 4 for an illustration of step 1.
Step 2. To evaluate a left caterpillar tree let $A_1$ and $A_2$ be two SLPs where $\text{val}(A_1) \in \{\land, \lor\}^*$, $\text{val}(A_2) \in \{0, 1\}^*$, and $|\text{val}(A_2)| = |\text{val}(A_1)| + 1$. Let $\varphi : \{\land, \lor\}^* \to \{0, 1\}^*$ be the homomorphism with $\varphi(\land) = 1$ and $\varphi(\lor) = 0$. Using binary search, we compute the largest position $i$ such that the length-$i$ suffix of $\text{val}(A_2)$ is equal to the length-$i$ prefix of $\varphi(\text{val}(A_1))$. If $i = |\text{val}(A_1)|$, then the value of $\text{val}(A_1)\text{val}(A_2)$ is the first symbol of $\text{val}(A_2)$. Otherwise, the value of $\text{val}(A_1)\text{val}(A_2)$ is 0 (resp., 1) if $\text{val}(A_1)[i + 1] = \land$ (resp., $\text{val}(A_1)[i + 1] = \lor$).

**Corollary 3.** If the interpretation $I$ is such that $(\mathcal{P}, \land^I, \lor^I)$ is a finite distributive lattice, then the $I$-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 $\{(0, 1), \land, \lor\}$.

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

In [2] it was shown that the problem of computing the $k$th bit ($k$ is given in binary notation) of the number to which a given arithmetical circuit evaluates to belongs to the counting hierarchy. An arithmetical 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 [15] that a TSLP $A$ over $+$ and $\times$ can be transformed in logspace into an arithmetical circuit that evaluates to $\text{val}(A)^I$. 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 $k$th bit of the output value of a TSLP-represented arithmetical expression belongs to the counting hierarchy. Here, we show that this result even holds for arithmetical expressions that are given by SLPs:

**Theorem 11.** The problem of computing for a given binary encoded number $k$ and an SLP $A$ over $\{+, \times\} \cup \mathbb{Z}$ the $k$th bit of $\text{val}(A)^I$ belongs to the counting hierarchy.

**Proof.** We follow the strategy from [2] proof of Thm. 4.1. Let $A$ be the input SLP for the tree $t$ and let $N = I(t)$. Then $N \leq 2^n$ where $n = |A|$ (this follows since the expression $t$ has size $2^n$ 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 $2^{2n} \geq \log^2 N$). Then $\prod_{p \in P_n} p > N$. Also note that each prime $p \in P_n$ has at most $2n$ bits in its binary representation. We first show that the language

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

belongs to the counting hierarchy. The rest of proof then follows the argument in [2]: Using the $\text{DLOGTIME}$-uniform $\text{TC}^0$-circuit family from [13] for transforming a number from its Chinese remainder representation into its binary representation one defines a $\text{TC}^0$-circuit of size $2^{O(n)}$ that has input gates $x(p, j)$ (where $n = |A|$, $p \in P_n$, $1 \leq j \leq 2n$). If we set $x(p, j)$ to true iff $(A, p, j) \in L$ (this means that the input gates $x(p, j)$ receive the Chinese remainder representation of $\text{val}(A)^I$), then the circuit outputs correctly the (exponentially many) bits of the binary representation of $\text{val}(A)^I$. Then, as in [2] 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 $O(n)$) 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 [2] proof of Thm. 4.1: In [2], the arithmetical expression is given by a circuit instead of an SLP. This simplifies the proof, because if we replace in the above language $L$ the
Hence we have to show that $L$ belongs to the counting hierarchy. Let $\mathcal{A}$ be an SLP for a tree
$t, n = |\mathcal{A}|, p \in P_n$, and $1 \leq j \leq 2n$. By Theorem \[ it suffices to consider the case that $t$ is a
caterpillar tree $t$; the polynomial time Turing reduction in Theorem \[ increases the level by one.
Also note that we use a uniform version of Theorem \[ 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 $\text{val}(\mathcal{A})$. Note that $m$ can be exponentially large in $|\mathcal{A}|$, but its binary representation
can be computed in polynomial time by Lemma \[ (point \[ ). We now define a matrix of numbers
$x_{i,j}^t \in \mathbb{Z}_p \ (i, j \in [1, m+1])$ such that
\[
x^t = \sum_{i=1}^{m+1} \prod_{j=1}^{m+1} x_{i,j}^t.
\]
Moreover, we will show that given $\mathcal{A}$ and binary encoded numbers $i, j \in [1, m+1]$, the binary
encoding of $x_{i,j}^t$ (which consists of at most $2n$ bits) can be computed in polynomial time.

We define the numbers $x_{i,j}^t$ inductively over the structure of the caterpillar tree $t$. For the
caterpillar tree $t = a$ (with $a \in \mathbb{Z}_p$) we set $x_{1,1}^t = a$. 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
$t = f(a, b)$ 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 $x_{i,j}^s$ are already defined for $i, j \in [1, m]$. If $f = +$, then we set
\[
x_{1,1}^t = a,
x_{1,j}^t = 1 \text{ for } i \in [2, m+1],
x_{i,1}^t = 1 \text{ for } i \in [2, m+1],
x_{i,j}^t = x_{i-1,j-1}^s \text{ for } i, j \in [2, m+1],
\]
We get
\[
\sum_{i=1}^{m+1} \prod_{j=1}^{m+1} x_{i,j}^t = a + \sum_{i=2}^{m+1} \prod_{j=2}^{m+1} x_{i-1,j-1}^s = a + \sum_{i=1}^{m} \prod_{j=1}^{m} x_{i,j}^s = a + s^t = t^2.
\]
If $f = \times$, then we set
\[
x_{1,i}^t = 0 \text{ for } i \in [1, m+1],
x_{i,1}^t = a \text{ for } i \in [2, m+1],
x_{i,j}^t = x_{i-1,j-1}^s \text{ for } i, j \in [2, m+1],
\]
We get
\[
\sum_{i=1}^{m+1} \prod_{j=1}^{m+1} x_{i,j}^t = a \sum_{i=2}^{m+1} \prod_{j=2}^{m+1} x_{i-1,j-1}^s = a \sum_{i=1}^{m} \prod_{j=1}^{m} x_{i,j}^s = a \cdot s^t = t^2.
\]

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

- If $t = a \in \mathbb{Z}_p$, then let operand($t, 1) = a$ (note that in this case we have $m = 0$, hence the
op($t, i$) do not exist).

SLP $\mathcal{A}$ 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.
A problem belongs to the counting hierarchy. We have shown that where the binary encoding of the number $x$ be computed in polynomial time using point (b) of Theorem 5. It produces a caterpillar tree, and the sum (resp., product) of $\Gamma$ is possible in polynomial time: The position $op(t, i)$ can be computed in polynomial time. This allows (again no effort to compute it.)(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 11 to show the following related result.

We have shown that

\[ val(\mathcal{A})^p = \sum_{i=1}^{m+1} \prod_{j=1}^{m+1} x_{i,j}^t. \]

where the binary encoding of the number $x_{i,j} \in \mathbb{Z}_p$ can be computed in polynomial time, given $\mathcal{A}, i, j.$ We now follow again the arguments from [2]. It is known that the binary representation of a sum (resp., product) of many $n$-bit numbers can be computed in $\text{DLOGTIME}$-uniform $\text{TC}^0$ [13]. 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 $O(\log n)$ bits (it is actually much easier to argue that the latter problem is in $\text{DLOGTIME}$-uniform $\text{TC}^0$; see again [13]). Hence, there is a $\text{DLOGTIME}$-uniform $\text{TC}^0$ circuit family $(C_m)_{m \geq 1},$ where the input of $C_m$ consists of bits $x(i, j, k) (i, j \in [1, m], k \in O(\log m))$ and a prime number $p$ with $O(\log m)$ bits, such that the following holds: If $x(i, j, k)$ receives the $k^{th}$ bit of a number $x_{i,j} \in \mathbb{Z}_p,$ then the circuit outputs $\sum_{i=1}^{m} \prod_{j=1}^{m+1} x_{i,j} \mod p.$ We take the circuit $C_{m+1},$ where $m \in 2^{O(n)}$ (recall that $n = |\mathcal{A}|$ and $m$ is the number of operators in $t = val(\mathcal{A}).$) The input gate $x(i, j, k)$ receives the $k^{th}$ bit of the number $x_{i,j} \in \mathbb{Z}_p$ defined above. We have shown above that the bits of $x_{i,j}$ can be computed in polynomial time. This allows (again in the same way as in [2] 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 arithmetical circuit belongs to $\text{PH}^{\text{PP}^\text{pnp}}$ [11] (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 11 to show the following related result.

\[ \text{The problem of evaluating SLP-compressed } \{\text{max}, +\} \cup \mathbb{Z} \text{-trees over the integers belongs to the counting hierarchy.} \]

**Proof.** The proof follows the arguments from the proof of Theorem [11]. 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 [11] and can use Theorem [7]
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 13 and define numbers \(x'_{i,j}\) (where \(t = \text{val}(A)\) is the input caterpillar tree) such that

\[
\text{val}(A)^2 = \max_{1 \leq i \leq m+1} \sum_{j=1}^{m+1} x'_{i,j}.
\]

Since the sum of \(n\) many \(n\)-bit numbers as well as the maximum of \(n\) many \(n\)-bit numbers can be computed in \(\text{DLOGTIME}\)-uniform \(\text{TC}^0\) (the maximum of \(n\) many \(n\)-bit numbers can be even computed in \(\text{DLOGTIME}\)-uniform \(\text{AC}^0\)), one can argue as in the proof of Theorem 13. \(\Box\)

Let us now turn to lower bounds for the problems of evaluating SLP-compressed arithmetical expressions (max-plus or plus-times). For a number \(c \in \mathbb{N}\) consider the unary operation \(+c\) on \(\mathbb{N}\) with \(+c(z) = z + c\). By Theorem 8 the evaluation of SLP-compressed \((\{\max, +c\} \cup \mathbb{N})\)-trees is possible in polynomial time. The following theorem shows that the general case of SLP-compressed \((\{\max, +\} \cup \mathbb{N})\)-trees is more complicated.

**Theorem 13.** Evaluating SLP-compressed \((\{\max, +\} \cup \mathbb{N})\)-trees is \#P-hard.

**Proof.** Let \(A, B\) be two SLPs over \(\{0, 1\}\) with \(|\text{val}(A)| = |\text{val}(B)|\). We will reduce from the problem of counting the number of occurrences of \((1, 1)\) in the convolution \(\text{val}(A) \otimes \text{val}(B) \in \{\{0, 1\}^2\}^*\), which is known to be \#P-complete by [19]. Let \(\rho : \{0, 1\}^* \rightarrow \{\max, +\}^*\) be the homomorphism defined by \(\rho(0) = \max, \rho(1) = +\). One can compute in polynomial time from \(A\) and \(B\) an SLP for the tree \(\rho(\text{val}(A)) \cdot \text{rev}(\text{val}(B))\). The corresponding tree over \(\{\max, +, 0, 1\}\) evaluates to one plus the number of occurrences of \((1, 1)\) in the convolution \(\text{val}(A) \otimes \text{val}(B)\). \(\Box\)

In [2] it was shown that the computation of a certain bit of the output value of an arithmetical circuit (over \(+\) and \(\times\)) is \#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 [6], also the problem of computing a certain bit of \(\text{val}(A)^2\) for a given SLP \(A\) is \#P-hard. For the related problem PosSLP of deciding, whether a given arithmetical circuit computes a positive number, no non-trivial lower bound is known. For SLPs, the corresponding problem becomes PP-hard:

**Theorem 14.** The problem of deciding whether \(\text{val}(A)^2 \geq 0\) for a given SLP \(A\) over \(\{+, \times\} \cup \mathbb{Z}\) is PP-hard.

**Proof.** By [19], the following problem is PP-complete: Given SLPs \(A, B\) over \(\{0, 1\}\) where \(|\text{val}(A)| = |\text{val}(B)|\), and a binary encoded number \(z\), is the number of occurrences of \((1, 1)\) in the convoluted string \(\text{val}(A) \otimes \text{val}(B)\) at least \(z\)? We modify the proof of Theorem 13. Let \(A, B\) be SLPs over \(\{0, 1\}\), where \(N = |\text{val}(A)| = |\text{val}(B)|\). Pick \(n \geq 0\) such that \(2^n > 2N\). Let \(\rho_A : \{0, 1\}^* \rightarrow \{+, \times\}^*\) be the homomorphism defined by \(\rho_A(0) = +, \rho_A(1) = \times\), \(\rho_B : \{0, 1\}^* \rightarrow \{1, 2\}^*\) be the homomorphism defined by \(\rho_B(0) = 1, \rho_B(1) = 2\). One can compute in polynomial time from \(A\) and \(B\) an SLP for the tree \(\rho_A(\text{val}(A)) \cdot (2^n) \cdot \rho_B(\text{rev}(\text{val}(B)))\) (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 \(\text{val}(A) \otimes \text{val}(B)\) 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 by 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 \(R - 2^{n+z}\). \(\Box\)
Tree automata. (Bottom-up) tree automata (see [9] 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. This correspondence only holds for deterministic tree automata. On the other hand every nondeterministic tree automaton can be transformed into a deterministic one using a powerset construction. Formally, a nondeterministic (bottom-up) tree automaton \( A = (Q, F, \Delta, F) \) consists of a finite set of states \( Q \), a ranked alphabet \( F \), a set \( \Delta \) of transition rules of the form \( f(q_1, \ldots, q_n) \to q \) where \( f \in F_n \) and \( q_1, \ldots, q_n, q \in Q \), and a set of final states \( F \subseteq Q \).

A tree \( t \in T(F) \) is accepted by \( A \) if \( t \xrightarrow{\Delta} q \) for some \( q \in F \) where \( \xrightarrow{\Delta} \) is the rewriting relation defined by \( \Delta \) as usual. The uniform membership problem for tree automata asks whether a given tree automaton \( A \) accepts a given tree \( t \in T(F) \). In [18] it was shown that this problem is complete for the class \( \text{LogCFL} \), which is the closure of the context-free languages under logspace reductions. \( \text{LogCFL} \) is contained in \( \text{P} \) and \( \text{DSPACE}(\log^2(n)) \). For every fixed tree automaton, the membership problem belongs to \( \text{NC}^1 \) [18]. If the input tree is given by a TSLP, the uniform membership problem becomes \( \text{P-complete} \) [23]. For non-linear TSLPs (where a parameter may occur several times in a right-hand side) the uniform membership problem becomes \( \text{PSPACE-complete} \), and \( \text{PSPACE-hardness} \) holds already for a fixed tree automaton [21]. The same complexity bound holds for SLP-compressed trees (which in contrast to non-linear TSLPs only allow exponential compression):

**Theorem 15.** Given a tree automaton \( A \) and an SLP \( \mathcal{A} \) for a tree \( t \in T(F) \), it is \( \text{PSPACE} \)-complete to decide whether \( A \) accepts \( t \). \( \text{PSPACE} \)-hardness already holds for a fixed tree automaton.

**Proof.** For the upper bound we use the following lemma from [24]: If a function \( f: \Sigma^* \to \Gamma^* \) is \( \text{PSPACE} \)-computable and \( L \subseteq \Gamma^* \) belongs to \( \text{NSPACE}(\log^k(n)) \) for some constant \( k \), then \( f^{-1}(L) \) belongs to \( \text{PSPACE} \). Given an SLP \( \mathcal{A} \) for the tree \( t = \text{val}(\mathcal{A}) \), one can compute the tree \( t \) by a \( \text{PSPACE} \)-transducer by computing the symbol \( t[i] \) for every position \( i \in \{1, \ldots, |t|\} \). The current position can be stored in polynomial space and every query can be performed in polynomial time.

As remarked above the uniform membership problem for explicitly given trees can be solved in \( \text{DSPACE}(\log^2(n)) \).

For the lower bound we use a fixed regular language \( L \subseteq ((\{0,1\}^2)^*) \) from [19] such that the following problem is \( \text{PSPACE} \)-complete: Given two SLPs \( \mathcal{A} \) and \( \mathcal{B} \) over \( \{0,1\} \) with \( |\text{val}(\mathcal{A})| = |\text{val}(\mathcal{B})| \), is \( \text{val}(\mathcal{A}) \otimes \text{val}(\mathcal{B}) \in L \)?

Let \( A = (Q, \{0,1\}^2, \Delta, q_0, F) \) be a finite word automaton for \( L \). Let \( \mathcal{A}, \mathcal{B} \) be two SLPs over \( \{0,1\} \) with \( |\text{val}(\mathcal{A})| = |\text{val}(\mathcal{B})| \) and let \( T \) be an SLP for the comb tree \( t(u, v) \) where \( u = \text{rev}(\text{val}(\mathcal{A})) \) and \( v = \text{rev}(\text{val}(\mathcal{B})) \). We transform \( A \) into a tree automaton \( A_T \) over \( \{0, 1, 0, 1, 8\} \) with the state set \( Q \cup \{p_0, p_1\} \), the set of final states \( F \) and the following transitions:

\[
\begin{align*}
\$ & \to q_0, \\
0 & \to p_0, \\
1 & \to p_1, \quad \text{for } i \in \{0, 1\}, \\
f_i(q, p_j) & \to q', \quad \text{for } (q, (i, j), q') \in \Delta
\end{align*}
\]

The automaton \( A \) accepts the convolution \( \text{val}(\mathcal{A}) \otimes \text{val}(\mathcal{B}) \) if and only if the tree automaton \( A_T \) accepts \( t(u, v) \). \( \square \)

The \( \text{PSPACE} \)-hardness result in Theorem 15 can also be interpreted as follows: There exists a fixed finite algebra for which the evaluation problem for SLP-compressed trees is \( \text{PSPACE} \)-complete. This is a bit surprising if we compare the situation with dags or TSLP-compressed trees. For these, membership for tree automata is still doable in polynomial time [23], whereas the evaluation problem of arithmetical 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 arithmetical expressions (\( \text{PSPACE} \) 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. E. Allender, N. Balaji, and S. Datta. Low-depth uniform threshold circuits and the bit-complexity of straight line programs. In Proceedings of the 39th International Symposium of Mathematical Foundations of Computer Science, MFCS 2014, Part II, volume 8635 of Lecture Notes in Computer Science, pages 13–24. Springer, 2014.
2. 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.
3. E. Allender and K. W. Wagner. Counting hierarchies: Polynomial time and constant. Bulletin of the EATCS, 40:182–194, 1990.
4. A. Bertoni, C. Choffrut, and R. Radicioni. Literal shuffle of compressed words. In Proceedings of the 5th IFIP International Conference on Theoretical Computer Science, TCS 2008, IFIP 20th World Computer Congress, volume 273 of IFIP, pages 87–100. Springer, 2008.
5. M. Bousquet-Mélou, M. Lohrey, S. Maneth, and E. Noeth. XML compression via DAGs. Theory of Computing Systems, 2014.
6. G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. Information Systems, 33(4–5):456–474, 2008.
7. H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC$^1$ computation. Journal of Computer and System Sciences, 57(2):200–212, 1998.
8. 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.
9. 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/.
10. J. Esparza, M. Luttenberger, and M. Schlund. A brief history of strahler numbers. In Proceedings of the 8th International Conference on Language and Automata Theory and Applications, LATA 2014, volume 8370 of Lecture Notes in Computer Science, pages 1–13. Springer, 2014.
11. P. Flajolet, J.-C. Raoult, and J. Vuillemin. The number of registers required for evaluating arithmetic expressions. Theoretical Computer Science, 9:99–125, 1979.
12. C. Hagenah. Gleichungen mit regulären Randbedingungen über freien Gruppen. PhD thesis, University of Stuttgart, Institut für Informatik, 2000.
13. 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.
14. D. Hucke, M. Lohrey, and E. Noeth. Constructing small tree grammars and small circuits for formulas. In Proceedings of the 34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, volume 29 of LIPIcs, pages 457–468. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.
15. D. Hucke, 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
16. A. Jez. Approximation of grammar-based compression via recompression. In Proceedings of the 24th Annual Symposium on Combinatorial Pattern Matching, CPM 2013, volume 7922 of Lecture Notes in Computer Science, pages 165–176. Springer, 2013.
17. A. Jez and M. Lohrey. Approximation of smallest linear tree grammars. In Proceedings of 32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2014, volume 25 of LIPIcs, pages 445–457. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.
18. M. Lohrey. On the parallel complexity of tree automata. In *Proceedings of the 12th International Conference on Rewriting Techniques and Applications, RTA 2001*, volume 2051 of *LNCS*, pages 201–215. Springer, 2001.

19. M. Lohrey. Leaf languages and string compression. *Information and Computation*, 209(6):951–965, 2011.

20. M. Lohrey. *The Compressed Word Problem for Groups*. Springer, 2014.

21. M. Lohrey and S. Maneth. The complexity of tree automata and XPath on grammar-compressed trees. *Theoretical Computer Science*, 363(2):196–210, 2006.

22. M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using RePair. *Information Systems*, 38(8):1150–1167, 2013.

23. 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.

24. M. Lohrey and C. Mathissen. Isomorphism of regular trees and words. *Information and Computation*, 224:71–105, 2013.

25. W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302(1–3):211–222, 2003.

26. H. Sakamoto. A fully linear-time approximation algorithm for grammar-based compression. *Journal of Discrete Algorithms*, 3(2-4):416–430, 2005.

27. M. Schmidt-Schauß. Linear compressed pattern matching for polynomial rewriting (extended abstract). In *Proceedings of the 7th International Workshop on Computing with Terms and Graphs, TERMGRAPH 2013*, volume 110 of *EPTCS*, pages 29–40, 2013.