Constructing small tree grammars and small circuits for formulas

Danny Hucke, Markus Lohrey, and Eric Noeth

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

Abstract

It is shown that every tree of size n over a fixed set of σ different ranked symbols can be decomposed into $O(\frac{n}{\log_\sigma n}) = O(\frac{n \log \sigma}{\log n})$ many hierarchically defined pieces. Formally, such a hierarchical decomposition has the form of a straight-line linear context-free tree grammar of size $O(\frac{n}{\log_\sigma n})$, which can be used as a compressed representation of the input tree. This generalizes an analogous result for strings. Previous grammar-based tree compressors were not analyzed for the worst-case size of the computed grammar, except for the top dag of Bille et al., for which only the weaker upper bound of $O(\frac{n}{\log^{0.19} n})$ for unranked and unlabelled trees has been derived. The main result is used to show that every arithmetical formula of size n, in which only $m \leq n$ different variables occur, can be transformed (in time $O(n \log n)$) into an arithmetical circuit of size $O(\frac{n \cdot \log m}{\log n})$ and depth $O(\log n)$. This refines a classical result of Brent, according to which an arithmetical formula of size n can be transformed into a logarithmic depth circuit of size $O(n)$. Missing proofs can be found in the long version [14].

1998 ACM Subject Classification E.4 Data compaction and compression

Keywords and phrases grammar-based compression, tree compression, arithmetical circuits

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

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*, briefly 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 grammar is much smaller than the size (length) of the string $(ab)^{1024}$. In general, an SLP of size n (the size of an SLP is usually defined as the total length of all right-hand sides of the productions) 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 analyzed. Prominent grammar-based string compressors are for instance LZ78, RePair, and BISECTION, see [7] for more details.

To evaluate the compression performance of a grammar-based compressor $\mathcal{C}$, two different approaches can be found in the literature: A first approach is to analyze the size of the SLP produced by $\mathcal{C}$ for an input string x compared to the size of a smallest SLP for x. This leads to the approximation ratio for $\mathcal{C}$, see [7] for a formal definition. It is known that unless $\mathsf{P} = \mathsf{NP}$, there is no polynomial time grammar-based compressor that produces for every string x an SLP of size strictly smaller than $8569/8568 \cdot g(x)$, where $g(x)$ is the size of a smallest SLP for x [7]. The best known polynomial time grammar-based compressors have

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

an approximation ratio of $\mathcal{O}(\log(n/g))$, where g is the size of a smallest SLP for the input string, and each of them works in linear time; see [22] for references.

Another approach is to analyze the maximal size of SLPs produced by $\mathcal{C}$ on strings of length n over the alphabet Σ (the size of Σ is considered to be a constant larger than one in the further discussion). An information-theoretic argument shows that for almost all strings of length n (up to an exponentially small part) the smallest SLP has size $\Omega(\frac{n}{\log n})$. Explicit examples of strings for which the smallest SLP has size $\Omega(\frac{n}{\log n})$ result from de Bruijn sequences; see Section 2. On the other hand, many grammar-based compressors produce for every string of length n an SLP of size $O(\frac{n}{\log n})$. This holds for instance for the above mentioned LZ78, RePair, and BISECTION, and in fact for all compressors that produce so-called irreducible SLPs [16]. This fact is used in [16] to construct universal string compressors based on grammar-based compressors.

In this paper, we follow the latter approach, but for trees instead of strings. 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. In [6], grammar-based compression was extended from strings to trees. For this, linear context-free tree grammars were used. Linear context-free tree grammars that produce only a single tree are also known as tree straight-line programs (TSLPs) or straight-line context-free tree grammars (SLCF tree grammars). 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.

Several grammar-based tree compressors were developed in [1, 6, 15, 23]. The algorithm from [15] achieves an approximation ratio of $O(\log n)$ (for a constant set of node labels). On the other hand, for none of the above mentioned compressors it is known, whether for any input tree with n nodes the size of the output grammar is bounded by $O(\frac{n}{\log n})$, as it is the case for many grammar-based string compressors. Recently, it was shown that the so-called *top dag* of an unranked and unlabelled tree of size n has size $O(\frac{n}{\log^{0.19} n})$ [3]. The top dag can be seen as a slight variant of a TSLP for an unranked tree.

In this paper, we present a grammar-based tree compressor that transforms a given node-labelled ranked tree of size n with σ different node labels into a TSLP of size $O(\frac{n}{\log_\sigma n})$ and depth $O(\log n)$, where the depth of a TSLP is the depth of the corresponding derivation tree. In particular, for an unlabelled binary tree we get a TSLP of size $O(\frac{n}{\log n})$. Our compressor is an extension of the BISECTION algorithm [17] from strings to trees and works in two steps (the following outline works only for binary trees, but it can be easily adapted to trees of higher ranks): In the first step, we hierarchically decompose the tree into pieces of roughly equal size, using a well-known lemma from [19]. But care has to be taken to bound the ranks of the nonterminals of the resulting TSLP. As soon as we get a tree with three holes during the decomposition (which corresponds in the TSLP to a nonterminal of rank three) we do an intermediate step that decomposes the tree into two pieces having only two holes each. This may involve an unbalanced decomposition. On the other hand, such an unbalanced decomposition is only necessary in every second step. This trick to bound the number of holes by three was used by Ruzzo [25] in his analysis of space-bounded alternation.

The TSLP produced in the first step can be identified with its derivation tree. Thanks to the fact that all nonterminals have rank at most three, we can encode the derivation tree by a tree with $O(\sigma)$ many labels. Moreover, this derivation tree is weakly balanced in the following sense. For each edge (u, v) in the derivation tree such that both u and v are internal nodes, the derivation tree is balanced at u or v. These facts allow us to show that the minimal dag of the derivation tree has size at most $O(\frac{n}{\log_\sigma n})$. The nodes of this dag are

the nonterminals of our final TSLP. The running time of our algorithm is in $O(n \log n)$.

Our size bound $O(\frac{n}{\log_\sigma n})$ does not contradict the information-theoretic lower bound: Consider for instance unlabelled ordered trees. When encoding a TSLP of size m into a bit string, we get an additional $\log(m)$-factor. Hence, a TSLP of size $O(\frac{n}{\log n})$ is encoded by a bit string of size $O(n)$, which is the information-theoretic bound (the exact bound is $2n - o(n)$).

It is important to note that our size bound $O(\frac{n}{\log_\sigma n})$ only holds for ranked trees and does not directly apply to unranked trees (that are, for instance, the standard tree model for XML). To overcome this limitation, one can transform an unranked tree of size n into its first-child-next-sibling encoding [18, Paragraph 2.3.2], which is a ranked tree of size n. Then, the first-child-next-sibling encoding can be transformed into a TSLP of size $O(\frac{n}{\log_\sigma n})$.

Our main result has an interesting application for the classical problem of transforming formulas into small circuits. Spira [26] has shown that for every Boolean formula of size n there exists an equivalent Boolean circuit of depth $O(\log n)$ and size $O(n)$. Brent [4] extended Spira's theorem to formulas over arbitrary semirings and moreover improved the constant in the $O(\log n)$ bound. Subsequent improvements that mainly concern constant factors can be found in [5]. An easy corollary of our $O(\frac{n}{\log_\sigma n})$ bound for TSLPs is that for every (not necessarily commutative) semiring (or field), every formula of size n, in which only $m \leq n$ different variables occur, can be transformed into a circuit of depth $O(\log n)$ and size $O(\frac{n \cdot \log m}{\log n})$. Hence, we refine the size bound from $O(n)$ to $O(\frac{n \cdot \log m}{\log n})$ (Theorem 9). Another interesting point of our formula-to-circuit conversion is that most of the construction (namely the construction of a TSLP for the input formula) is purely syntactic. The remaining part (the transformation of the TSLP into a circuit) is straightforward.

Related work. Several papers deal with algorithmic problems on trees that are succinctly represented by TSLPs, see [22] for a survey. Among other problems, equality checking and the evaluation of tree automata can be done in polynomial time for TSLPs.

It is interesting to compare our $O(\frac{n}{\log_\sigma n})$ bound with the known bounds for dag compression. A counting argument shows that for almost all unlabelled binary trees, the size of a smallest TSLP is $\Omega(\frac{n}{\log n})$, and hence (by our main result) $\Theta(\frac{n}{\log n})$. This implies that the average size of the minimal TSLP, where the average is taken for the uniform distribution on unlabelled binary trees of size n, is $\Theta(\frac{n}{\log n})$ as well. In contrast, the size of the minimal dag for trees of size n is $\Theta(n/\sqrt{\log n})$ on average [11] but n in the worst case.

2 Strings and Straight-Line Programs

Before we come to grammar-based tree compression, let us briefly discuss grammar-based string compression. A *straight-line program*, briefly SLP, is a context-free grammar that produces a single string. Formally, it is a tuple $\mathcal{G} = (N, \Sigma, P, S)$, where N is a finite set of nonterminals, Σ 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 of the form $A \to w$ for $A \in N$, $w \in (N \cup \Sigma)^*$ such that: (i) if $(A \to u), (A \to v) \in P$ then $u = v$, and (ii) the binary relation $\{(A, B) \in N \times N \mid (A \to w) \in P,\ B \text{ occurs in } w\}$ is acyclic. Every nonterminal $A \in N$ produces a unique string $\mathrm{val}_\mathcal{G}(A) \in \Sigma^*$. The string defined by $\mathcal{G}$ is $\mathrm{val}(\mathcal{G}) = \mathrm{val}_\mathcal{G}(S)$. The size of the SLP $\mathcal{G}$ is $|\mathcal{G}| = \sum_{(A \to w) \in P} |w|$, where $|w|$ is the length of w.

Let σ be the size of the terminal alphabet Σ. It is well-known that for every string $x \in \Sigma^*$ of length n there exists an SLP $\mathcal{G}$ of size $O(n/\log_\sigma n)$ such that $\mathrm{val}(\mathcal{G}) = x$, see e.g. [16]. On the other hand, an information-theoretic argument shows that for almost all strings of length n, the smallest SLP has size $\Omega(n/\log_\sigma n)$. For SLPs, one can, in contrast to other models like Boolean circuits, construct explicit strings that achieve this worst-case bound:

▶ **Proposition 1.** *Let Σ be an alphabet of size σ. For every $n \geq \sigma^2$, one can construct in time* $\mathrm{poly}(n, \sigma)$ *a string $s_{\sigma,n} \in \Sigma^*$ of length n such that every SLP for $s_{\sigma,n}$ has size $\Omega(n/\log_\sigma n)$.*

Proof. Let $r = \lceil \log_\sigma n \rceil \geq 2$. The sequence $s_{\sigma,n}$ is in fact a prefix of a de Bruijn sequence [9]. Let $x_1, \dots x_{\sigma^{r-1}}$ be a list of all words from Σ^{r-1}. Construct a directed graph by taking these strings as vertices and drawing an a-labelled edge ($a \in \Sigma$) from x_i to x_j if $x_i = bw$ and $x_j = wa$ for some $w \in \Sigma^{r-2}$ and $b \in \Sigma$. This graph has σ^r edges and every vertex of this graph has indegree and outdegree σ. Hence, it has a Eulerian cycle, which can be viewed as a sequence $u, b_1, b_2, \dots, b_{\sigma^r}$, where $u \in \Sigma^{r-1}$ is the start vertex, and the edge traversed in the i^{th} step is labelled with $b_i \in \Sigma$. Define $s_{\sigma,n}$ as the prefix of $ub_1b_2 \cdots b_{\sigma^r}$ of length n. The construction implies that $s_{\sigma,n}$ has $n - r + 1$ different substrings of length r. By the so-called mk-Lemma from [7], every SLP for $s_{\sigma,n}$ has size at least $\frac{n-r+1}{r} > \frac{n}{r} - 1 \geq \frac{n}{\log_\sigma(n)+1} - 1$. ◀

In [2] a set of n binary strings of length n is constructed such that any concatenation circuit that computes this set has size $\Omega(n^2/\log^2 n)$. A concatenation circuit for a set S of strings is simply an SLP such that every string from S is derived from a nonterminal of the SLP. Using the above construction, this lower bound can be improved to $\Omega(n^2/\log n)$: Simply take the string s_{2,n^2} and write it as $s_1 s_2 \cdots s_n$ with $|s_i| = n$. Then any concatenation circuit for $\{s_1, \dots, s_n\}$ has size $\Omega(n^2/\log n)$.

3 Trees and Tree Straight-Line Programs

For every $i \geq 0$, we fix a countably infinite set $\mathcal{F}_i$ (resp., $\mathcal{N}_i$) of *terminals* (resp., *nonterminals*) of rank i. Let $\mathcal{F} = \bigcup_{i \geq 0} \mathcal{F}_i$ and $\mathcal{N} = \bigcup_{i \geq 0} \mathcal{N}_i$. Moreover, let $\mathcal{X} = \{x_1, x_2, \dots\}$ be a countably infinite set of *parameters*. We assume that $\mathcal{F}$, $\mathcal{N}$, and $\mathcal{X}$ are pairwise disjoint. A *labelled tree* $t = (V, \lambda)$ is a finite, rooted and ordered tree t with node set V and labelling function $\lambda : V \to \mathcal{F} \cup \mathcal{N} \cup \mathcal{X}$. We require that a node $v \in V$ with $\lambda(v) \in \mathcal{F}_k \cup \mathcal{N}_k$ has exactly k children, which are ordered from left to right. We also require that every node v with $\lambda(v) \in \mathcal{X}$ is a leaf of t. The size of t is $|t| = |\{v \in V \mid \lambda(v) \in \mathcal{F} \cup \mathcal{N}\}|$, i.e., we do not count parameters. We denote trees in their usual term notation, e.g. $b(a, a)$ denotes the tree with a b-labelled root, which has two a-labelled children. We define $\mathcal{T}$ as the set of all labelled trees. The *depth* of a tree t is the maximal length (number of edges) of a path from the root to a leaf, and is denoted by $\mathrm{depth}(t)$. Let $\mathrm{labels}(t) = \{\lambda(v) \mid v \in V\}$ and $\mathcal{T}(\mathcal{L}) = \{t \mid \mathrm{labels}(t) \subseteq \mathcal{L}\}$ for $\mathcal{L} \subseteq \mathcal{F} \cup \mathcal{N} \cup \mathcal{X}$. We write $<_t$ for the depth-first-order on V. Formally, $u <_t v$ if u is an ancestor of v or if there exists a node w and $i < j$ such that the i^{th} child of w is an ancestor of u and the j^{th} child of w is an ancestor of v. The tree $t \in \mathcal{T}$ is *linear* if there do not exist different nodes that are labelled with the same parameter. We call $t \in \mathcal{T}$ *valid* if (i) $\mathrm{labels}(t) \cap \mathcal{X} = \{x_1, \dots, x_n\}$ for some $n \geq 0$ and (ii) for all $u, v \in V$ with $\lambda(u) = x_i$, $\lambda(v) = x_j$ and $u <_t v$ we have $i < j$ (in particular t is linear). For example, $f(x_1, x_{21}, x_{99})$, $f(x_1, x_1, x_3)$, and $f(x_3, x_1, x_2)$ are invalid, whereas $f(x_1, x_2, x_3)$ is valid. For a linear tree t we define $\mathrm{valid}(t)$ as the unique valid tree which is obtained from t by renaming the parameters. For instance, $\mathrm{valid}(f(x_{21}, x_2, x_{99})) = f(x_1, x_2, x_3)$. A valid tree t in which the parameters $x_1, \dots, x_n$ occur is also written as $t(x_1, \dots, x_n)$ and we write $\mathrm{rank}(t) = n$.

We now define a particular form of context-free tree grammars (see [8] for more details on context-free tree grammars) with the property that exactly one tree is derived. A *tree straight-line program (TSLP)* is a pair $\mathcal{G} = (S, P)$, where $S \in \mathcal{N}_0$ is the start nonterminal and P is a finite set of rules of the form $A(x_1, \dots, x_n) \to t(x_1, \dots, x_n)$ (which is also briefly written as $A \to t$), where $n \geq 0$, $A \in \mathcal{N}_n$ and $t(x_1, \dots, x_n) \in \mathcal{T}$ is valid such that:

- There is an initial rule $(S \to t) \in P$.

- If $(A \to s) \in P$ and $B \in \text{labels}(s) \cap \mathcal{N}$, then there is a tree t such that $(B \to t) \in P$.
- There do not exist rules $(A \to t_1), (A \to t_2) \in P$ with $t_1 \neq t_2$.
- The binary relation $\{(A, B) \in \mathcal{N} \times \mathcal{N} \mid (A \to t) \in P, B \in \text{labels}(t)\}$ is acyclic.

These conditions ensure that from every nonterminal $A \in \mathcal{N}_n$ exactly one valid tree $\text{val}_{\mathcal{G}}(A) \in \mathcal{T}(\mathcal{F} \cup \{x_1, \ldots, x_n\})$ is derived by using the rules as rewrite rules in the usual sense. The tree defined by $\mathcal{G}$ is $\text{val}(\mathcal{G}) = \text{val}_{\mathcal{G}}(S)$. Instead of a formal definition, we give an example:

▶ **Example 2.** Let $\mathcal{G} = (S, P)$, where P consists of the following rules ($a \in \mathcal{F}_0$, $b \in \mathcal{F}_2$): $S \to A(B)$, $A(x_1) \to C(F, x_1)$, $B \to E(F)$, $C(x_1, x_2) \to D(E(x_1), x_2)$, $D(x_1, x_2) \to b(x_1, x_2)$, $E(x_1) \to D(F, x_1)$, $F \to a$. Part of a possible derivation of $\text{val}(\mathcal{G}) = b(b(a, a), b(a, a))$ from S is: $S \to A(B) \to C(F, B) \to D(E(F), B) \to b(E(F), B) \to b(D(F, F), B) \to b(b(F, F), B) \to b(b(a, F), B) \to b(b(a, a), B) \to b(b(a, a), E(F)) \to \cdots \to b(b(a, a), b(a, a))$.

The size $|\mathcal{G}|$ of a TSLP $\mathcal{G} = (S, P)$ is the total size of all trees on the right-hand sides of P: $|\mathcal{G}| = \sum_{(A \to t) \in P} |t|$. For instance, the TSLP from Example 2 has size 12.

A TSLP is in *Chomsky normal form* if for every production $A(x_1, \ldots, x_n) \to t(x_1, \ldots, x_n)$ one of the following two cases holds:

$$t(x_1, \ldots, x_n) = B(x_1, \ldots, x_{i-1}, C(x_i, \ldots, x_k), x_{k+1}, \ldots, x_n) \text{ for } B, C \in \mathcal{N} \quad (1)$$

$$t(x_1, \ldots, x_n) = f(x_1, \ldots, x_n) \text{ for } f \in \mathcal{F}_n. \quad (2)$$

If the tree t in the corresponding rule $A \to t$ is of type (1), we write $\text{index}(A) = i$. If otherwise t is of type (2), we write $\text{index}(A) = 0$. One can transform every TSLP efficiently into an equivalent TSLP in Chomsky normal form with a small size increase [24]. We only consider TSLPs in Chomsky normal form in the following.

We define the rooted, ordered derivation tree $\mathcal{D}_{\mathcal{G}}$ of a TSLP $\mathcal{G} = (S, P)$ in Chomsky normal form as for string grammars: The inner nodes of the derivation tree are labelled by nonterminals and the leaves are labelled by terminal symbols. Formally, we start with the root node of $\mathcal{D}_{\mathcal{G}}$ and assign it the label S. For every node in $D_{\mathcal{G}}$ labelled by A, where the right-hand side t of the rule for A is of type (1), we attach a left child labelled by B and a right child labelled by C. If the right-hand side t of the rule for A is of type (2), we attach a single child labelled by f to A. Note that these nodes are the leaves of $\mathcal{D}_{\mathcal{G}}$ and they represent the nodes of the initial tree $\text{val}(\mathcal{G})$. We denote by $\text{depth}(\mathcal{G})$ the depth of the derivation tree $\mathcal{D}_{\mathcal{G}}$. For instance, the depth of the TSLP from Example 2 is 4.

A commonly used compact tree compression scheme is obtained by writing down repeated subtrees only once. In that case all occurrences except for the first are replaced by a pointer to the first one. This leads to a node-labelled *directed acyclic graph* (dag). It is known that every tree has a unique minimal dag, which is called the *the dag* of the initial tree. An example can be found in Figure 2, where the right graph is the dag of the tree in the middle. The dag of a tree t can be constructed in time $O(|t|)$ [10]. Dags correspond to TSLPs where every nonterminal has rank 0.

4 Constructing a small TSLP for a tree

In this section we construct a TSLP $\mathcal{G}$ for a given tree t of size n. We then prove that $|\mathcal{G}| \in O(n/\log n)$. For the remainder of this section we restrict our input to binary trees, i.e., every node has either zero or two children. Formally, we consider trees from $\mathcal{T}(\mathcal{F}_0 \cup \mathcal{F}_2)$.

The following idea of splitting a tree recursively into smaller parts of roughly equal size is well-known, see e.g. [4, 26]. For our later analysis, it is important to bound the number of parameters in the resulting nonterminals (i.e., the number of holes in trees)

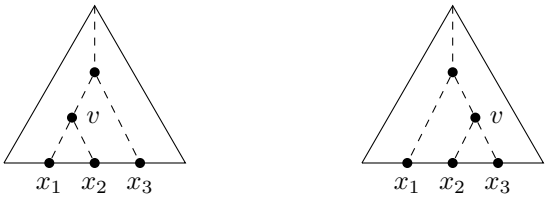


Figure 1 Splitting a tree with three parameters

by a constant. To achieve this, we use an idea from Ruzzo's paper [25]. For a valid tree $t = (V, \lambda) \in \mathcal{T}(\mathcal{F}_0 \cup \mathcal{F}_2 \cup \mathcal{X})$ and a node $v \in V$ we denote by $t[v]$ the tree valid(s), where s is the subtree rooted at v in t. We further write $t \setminus v$ for the tree valid(r), where r is obtained from t by replacing the subtree rooted at v by a new parameter. If for instance $t = h(g(x_1, f(x_2, x_3)), x_4)$ and v is the f-labelled node, then $t[v] = f(x_1, x_2)$ and $t \setminus v = h(g(x_1, x_2), x_3)$. The following lemma is well-known, see e.g. [19].

▶ **Lemma 3.** *Let t be a binary tree with $|t| \geq 2$. One can determine in time $O(|t|)$ a node v such that $\frac{1}{3}|t| - \frac{1}{2} \leq |t[v]| \leq \frac{2}{3}|t|$.*

For the remainder of this section we denote by split(t) the unique node in a tree t computed using Lemma 3. We now construct a TSLP $\mathcal{G}$ with val$(\mathcal{G}) = t$ for a given binary tree t (we assume that $|t| \geq 2$). Every nonterminal of $\mathcal{G}$ will be of rank at most three. We store two sets of productions, P_{temp} and P_{final}. The set P_{final} contains rules of the final TSLP $\mathcal{G}$ and P_{temp} ensures that the TSLP $(S, P_{\text{temp}} \cup P_{\text{final}})$ produces t at any point of time. Initially, we set $P_{\text{temp}} := \{S \to t\}$ and $P_{\text{final}} := \emptyset$. While P_{temp} is non-empty we proceed for each rule $(A \to s) \in P_{\text{temp}}$ as follows: Let $A \in \mathcal{N}_r$. If $r \leq 2$ we determine the node $v = \text{split}(s)$ in s. Then we split the tree s into the trees $s[v]$ and $s \setminus v$. Let $r_1 = \text{rank}(s[v])$, $r_2 = \text{rank}(s \setminus v)$ and let $A_1 \in \mathcal{N}_{r_1}$ and $A_2 \in \mathcal{N}_{r_2}$ be fresh nonterminals. Note that $r = r_1 + r_2 - 1$. If the size of $s[v]$ (resp., $s \setminus v$) is larger than 1 we add the rule $A_1 \to s[v]$ (resp., $A_2 \to s \setminus v$) to P_{temp}. Otherwise we add it to P_{final} as a final rule. Let k be the number of nodes of s that are labelled by a parameter and that are smaller (w.r.t. $<_s$) than v. To link the nonterminal A to the fresh nonterminals A_1 and A_2 we add the rule $A(x_1, \ldots, x_r) \to A_1(x_1, \ldots, x_k, A_2(x_{k+1}, \ldots, x_{k+r_2}), x_{k+r_2+1}, \ldots, x_r)$ to P_{final}.

To bound the rank of the nonterminals by three we handle rules $A \to s$ with $A \in \mathcal{N}_3$ as follows. Let v_1, v_2 and v_3 be the nodes labelled by the parameters x_1, x_2 and x_3, respectively. Instead of choosing the node v by split(s) we set v to the lowest common ancestor of (v_1, v_2) or (v_2, v_3), depending on which one has the greater distance from the root node (see Figure 1). This step ensures that the two trees $s[v]$ and $s \setminus v$ have rank 2, so in the next step each of them will be split in a balanced way according to Lemma 3. As a consequence, the resulting TSLP has depth $O(\log |t|)$ but size $O(|t|)$. The running time of this first phase can be bounded by $O(|t| \log |t|)$: All right-hand sides from P_{temp} obtained after i splittings have total size at most $|t|$, so we need time $O(|t|)$ to split them. Moreover, i ranges from 0 to $O(\log |t|)$.

▶ **Example 4.** If we apply our construction to the tree $b(b(a, a), b(a, a))$ we get the TSLP with the rules $S \to A(B)$, $A(x_1) \to C(D, x_1)$, $B \to E(F)$, $C(x_1, x_2) \to G(H(x_1), x_2)$, $D \to a$, $E(x_1) \to I(J, x_1)$, $F \to a$, $G(x_1, x_2) \to b(x_1, x_2)$, $H(x_1) \to K(L(x_1))$, $I(x_1, x_2) \to b(x_1, x_2)$, $J \to a$, $K(x_1, x_2) \to b(x_1, x_2)$, and $L \to a$.

In the next step we want to compact the TSLP by considering the dag of the derivation tree. For this we first build the derivation tree $\mathcal{D}_{\mathcal{G}}$ from the TSLP $\mathcal{G}$ as described above. The derivation tree for the TSLP described in Example 4 is shown on the left of Figure 2.

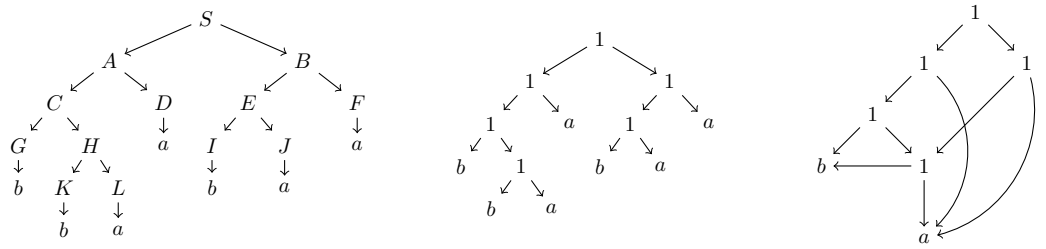


Figure 2 The derivation tree from Example 4

We now want to identify some (but not all) nonterminals that produce the same tree. Note that if we just omit the nonterminal labels from the derivation tree, then there might exist isomorphic subtrees of the derivation whose root nonterminals produce different trees. This is due to the fact that we lost for an A-labelled node of the derivation tree with a left (resp., right) child that is labelled with B (resp., C) the information at which argument position of B the nonterminal C is substituted. To keep this information we replace every label A in the derivation tree with $\mathrm{index}(A) \in \{0,1,2,3\}$ (the index of a nonterminal of a TSLP in Chomsky normal form was defined in Section 3). Moreover, we remove every leaf v and write its label into its parent node. We call the resulting tree the *modified derivation tree* and denote it by $\mathcal{D}^*_{\mathcal{G}}$. Note that $\mathcal{D}^*_{\mathcal{G}}$ is a full binary tree with node labels from $\{1,2,3\} \cup \mathrm{labels}(t)$. The modified derivation tree for Example 4 is shown in the middle of Figure 2. The following lemma shows how to compact our grammar by considering the dag of $\mathcal{D}^*_{\mathcal{G}}$.

▶ **Lemma 5.** *Let u and v be nodes of $\mathcal{D}_{\mathcal{G}}$ labelled by A resp. B. Moreover, let u' and v' be the corresponding nodes in $\mathcal{D}^*_{\mathcal{G}}$. If the subtrees $\mathcal{D}^*_{\mathcal{G}}[u']$ and $\mathcal{D}^*_{\mathcal{G}}[v']$ are isomorphic (as labelled ordered trees), then $\mathrm{val}_{\mathcal{G}}(A) = \mathrm{val}_{\mathcal{G}}(B)$.*

By Lemma 5, if two subtrees of $\mathcal{D}^*_{\mathcal{G}}$ are isomorphic we can eliminate the nonterminal of a root node of one subtree. Hence, we construct the dag d of $\mathcal{D}^*_{\mathcal{G}}$. This is possible in time $O(|\mathcal{D}_{\mathcal{G}}|) = O(|t|)$ [10]. The minimal dag of the TSLP of Example 4 is shown on the right of Figure 2. The nodes of d are the nonterminals of the final TSLP. We obtain rules of type (1) for each nonterminal corresponding to an inner node of d and rules of type (2) for each leaf in d. Let n_1 be the number of inner nodes of d and n_2 be the number of leaves. Then the size of our final TSLP is $2n_1 + n_2$, which is bounded by twice the number of nodes of d. The dag from Figure 2 gives the TSLP for the tree $b(b(a,a),b(a,a))$ described in Example 2.

To estimate the number of nodes in the dag of the modified derivation tree, we prove in this section a general result about the size of dags of certain weakly balanced binary trees. Let t be a binary tree and let $0 < \beta < 1$ and $\gamma \geq 2$ be constants. The *leaf size* of a node v is the number of leaves of the subtree rooted at v. We say that an inner node v with children v_1 and v_2 is *β-balanced* if the following holds: If n_i is the leaf size of v_i, then $n_1 \geq \beta n_2$ and $n_2 \geq \beta n_1$. We say that t is *(β,γ)-balanced* if the following holds: For all inner nodes u and v such that v is a child of u and the leaf size of v (and hence also u) is at least γ, we have that u is β-balanced or v is β-balanced.

▶ **Theorem 6.** *Fix constants $0 < \beta < 1$ and $\gamma \geq 2$. Then there is a constant α (depending on β and γ) such that the following holds: If t is a (β,γ)-balanced binary tree with n leaves and $|\mathrm{labels}(t)| = \sigma$ (hence, $|t|, \sigma \leq 2n-1$), then the size of the dag of t is bounded by $\frac{\alpha \cdot n}{\log_\sigma n}$.*

Proof. Let us fix a tree $t = (V,\lambda)$ as in the theorem with n leaves. Moreover, let us fix a number $k \geq \gamma$ that will be defined later. Let $\mathrm{top}(t,k)$ be the tree obtained from t by

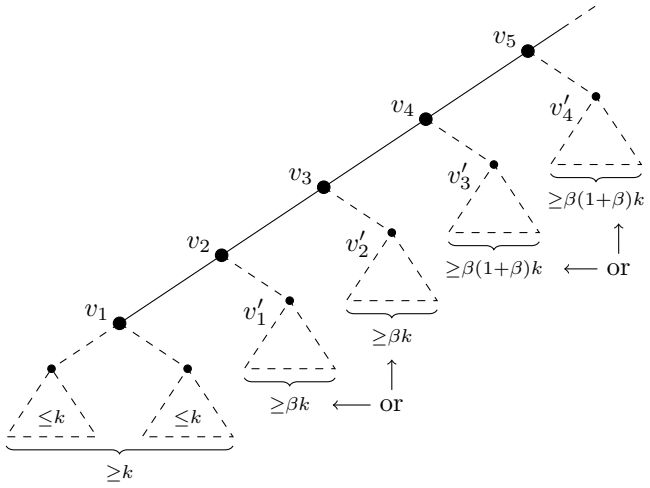


Figure 3 A chain within a top tree. The subtree rooted at v_1 has more than k leaves.

removing all nodes with leaf size at most k. We first bound the number of different subtrees with at most k leaves in t. Afterwards we will estimate the size of the remaining tree $\mathrm{top}(t,k)$. The same strategy is used for instance in [13, 20] to derive a worst-case upper bound on the size of binary decision diagrams.

Claim 1. The number of different subtrees of t with at most k leaves is bounded by d^k with $d = 4\sigma^2$.

A subtree of t with i leaves has exactly $2i-1$ nodes, each labelled with one of σ labels. Let $C_m = \frac{1}{m+1}\binom{2m}{m}$ be the m^{th} Catalan number. It is known that $C_m \leq 4^m$. If the labels are ignored, there are C_{i-1} different subtrees with i leaves. In conclusion, we get the following bound: $\sum_{i=1}^{k} C_{i-1}\cdot\sigma^{2i-1} \leq \sum_{i=0}^{k-1} 4^i\cdot\sigma^{2i+1} = \sigma\frac{(4\sigma^2)^k-1}{4\sigma^2-1} \leq (4\sigma^2)^k$.

Claim 2. The number of nodes of $\mathrm{top}(t,k)$ is bounded by $c\cdot\frac{n}{k}$ for a constant c depending only on β and γ.

The tree $\mathrm{top}(t,k)$ has at most n/k leaves since it is obtained from t by removing all nodes with leaf size at most k. Each node in $\mathrm{top}(t,k)$ has at most two children. Therefore it remains to show that the length of *unary chains* in $\mathrm{top}(t,k)$ is bounded by a constant.

Let $v_1,\ldots,v_m$ be a unary chain in $\mathrm{top}(t,k)$ where v_i is the single child node of v_{i+1}. Moreover, let v'_i be the removed sibling of v_i in t, see Figure 3. Note that each node v'_i has leaf size at most k. We claim that the leaf size of v_{2i+1} is larger than $(1+\beta)^i k$ for all i with $2i+1 \leq m$. For $i=0$ note that v_1 has leaf size more than k since otherwise it would have been removed in $\mathrm{top}(t,k)$. For the induction step, assume that the leaf size of v_{2i-1} is larger than $(1+\beta)^{i-1}k \geq k \geq \gamma$. One of the nodes v_{2i} and v_{2i+1} must be β-balanced. Hence, v'_{2i-1} or v'_{2i} must have leaf size more than $\beta(1+\beta)^{i-1}k$. Hence, v_{2i+1} has leaf size more than $(1+\beta)^{i-1}k + \beta(1+\beta)^{i-1}k = (1+\beta)^i k$.

Let $\ell = \log_{1+\beta}(\beta^{-1})$. If $m \geq 2\ell+3$, then $v_{2\ell+1}$ exists and has leaf size more than k/β, which implies that the leaf size of $v'_{2\ell+1}$ or $v'_{2\ell+2}$ (both nodes exist) is more than k, which is a contradiction. Hence, we must have $m \leq 2\log_{1+\beta}(\beta^{-1})+2$. Figure 3 shows an illustration.

Using Claim 1 and 2 we can now prove the theorem: The number of nodes of the dag of t is bounded by the number of different subtrees with at most k leaves (Claim 1) plus the number of nodes of the remaining tree $\mathrm{top}(t,k)$ (Claim 2). Let $k = \max\{\gamma, \frac{1}{2}\log_d n\} \geq \gamma$ (recall that $d = 4\sigma^2$ and hence $\log d = 2 + 2\log\sigma$). If $k = \gamma$, i.e., $\frac{1}{2}\log_d n \leq \gamma$ then we

have $\frac{n}{\log_\sigma n} \in \Omega(n)$ and the bound $O(\frac{n}{\log_\sigma n})$ on the size of the dag is trivial. If $k = \frac{1}{2}\log_d n$ then we get with Claim 1 and 2 the following bound on the size of the dag: $d^k + c \cdot \frac{n}{k} = d^{(\log_d n)/2} + 2c \cdot \frac{n}{\log_d n} = \sqrt{n} + 2c \cdot \frac{n}{\log_d n} \in O(\frac{n}{\log_d n}) = O(\frac{n}{\log_\sigma n})$. This proves the theorem. ◀

Obviously, one could relax the definition of (β, γ)-balanced by only requiring that if $(v_1, v_2, \ldots, v_\delta)$ is a path down in the tree, where δ is a constant and v_δ has leaf size at least γ, then one of the nodes $v_1, v_2, \ldots, v_\delta$ must be β-balanced. Theorem 6 would still hold with this definition (with the constant α also depending on δ).

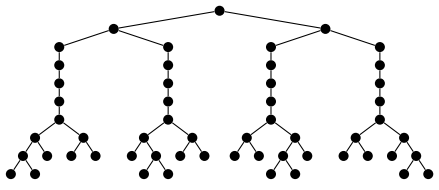
Let us fix the TSLP $\mathcal{G}$ for a binary tree $t \in \mathcal{T}(\mathcal{F}_0 \cup \mathcal{F}_2)$ that has been produced by the first part of our algorithm. Let $n = |t|$ and $\sigma = |\mathrm{labels}(t)|$. Then, the modified derivation tree $\mathcal{D}^*_\mathcal{G}$ is a binary tree with n leaves (and hence $2n-1$ nodes) and $\sigma + 3$ different node labels (namely $1, 2, 3$ and those appearing in t). Moreover, $\mathcal{D}^*_\mathcal{G}$ is $(1/3, 6)$-balanced: If we have two successive nodes in $\mathcal{D}^*_\mathcal{G}$, then we split at one of the two nodes according to Lemma 3. Now, assume that we split at node v according to Lemma 3. Let v_1 and v_2 be the children of v, let n_i be the leaf size of v_i, and let $n = n_1 + n_2 \geq 6$ be the leaf size of v. We get $\frac{1}{3}n - \frac{1}{2} \leq n_1 \leq \frac{2}{3}n$ and $\frac{1}{3}n \leq n_2 \leq \frac{2}{3}n + \frac{1}{2}$ (or vice versa). Since $n \geq 6$ we have $\frac{1}{4}n \leq n_1 \leq \frac{2}{3}n$ and $\frac{1}{3}n \leq n_2 \leq \frac{3}{4}n$. We get $n_1 \geq \frac{1}{4}n \geq \frac{1}{3}n_2$ and $n_2 \geq \frac{1}{3}n \geq \frac{1}{2}n_1$. Hence, we get:

▶ **Corollary 7.** *Let t be a binary tree with $|t| = n$ and $|\mathrm{labels}(t)| = \sigma$. Let d be the minimal dag of the modified derivation tree produced from t by our algorithm. Then the number of nodes of d is in $O(\frac{n}{\log_\sigma n})$. Hence, the size of the TSLP produced from t is in $O(\frac{n}{\log_\sigma n})$.*

The conditions in Theorem 6 ensure that $\mathrm{depth}(t) \in O(\log |t|)$. One might think that a tree t of depth $O(\log |t|)$ has a small dag. For instance, the dag of a complete binary tree with n nodes has size $O(\log n)$. But this intuition is wrong:

▶ **Theorem 8.** *There is a family of trees $t_n \in \mathcal{T}(\{a, c\})$ ($a \in \mathcal{F}_0$, $c \in \mathcal{F}_2$), $n \geq 1$, such that (i) $|t_n| \in O(n)$, (ii) $\mathrm{depth}(t) \in O(\log n)$, and (iii) the size of the dag of t_n is at least n.*

Proof. To simplify the presentation, we use a unary node label $b \in \mathcal{F}_1$. It can be replaced by the pattern $c(d, x)$, where $d \in \mathcal{F}_0 \setminus \{a\}$ to obtain a binary tree. Let $k = \frac{n}{\log n}$ (we ignore rounding problems with $\log n$, which only affects multiplicative factors). Choose k different binary trees $s_1, \ldots, s_k \in \mathcal{T}(\{a, c\})$, each having $\log n$ internal nodes. Note that this is possible since by the formula for the Catalan numbers there are more than n different binary trees with $\log n$ internal nodes for n large enough. Then consider the trees $s'_i = b^{\log n}(s_i)$. Each of these trees has size at most $3 \log n$ as well as depth at most $3 \log n$. Next, let $u_n(x_1, \ldots, x_k) \in \mathcal{T}(\{c, x_1, \ldots, x_k\})$ a binary tree (all non-parameter nodes are labelled with c) of depth $\log k \leq \log n$ and size $O(k) = O(\frac{n}{\log n})$. We finally take $t_n = u_n(s'_1, \ldots, s'_k)$. A possible choice for t_{16} is shown below. We obtain $|t_n| = O(\frac{n}{\log n}) + O(k \cdot \log n) = O(n)$. The depth of t_n is bounded by $3 \log n$. Finally, in the dag for t_n the unary b-labelled nodes cannot be shared. Basically, the pairwise different trees $t_1, \ldots, t_n$ work as different constants that are attached to the b-chains. But the number of b-labelled nodes in t_n is $k \cdot \log n = n$. ◀



It is straightforward to adapt our algorithm to trees where every node has at most r children for a fixed constant r. One only has to prove a version of Lemma 3 for r-ary trees. The multiplicative constant in the $O(\frac{n}{\log_\sigma n})$ bound for the final TSLP will depend on r. On the other hand, for unranked trees, where the number of children of a node is arbitrary, our

algorithm does not work. This problem can be solved by transforming an unranked tree into a binary tree of the same size using the first-child next-sibling encoding [18]. For this binary tree we get a TSLP of size $O\big(\frac{n}{\log_\sigma n}\big)$.

For traversing a compressed unranked tree t, another well-known encoding is favorable. Let c_t be a compressed representation (e.g., a TSLP) of t. The goal is to represent t in space $O(|c_t|)$ such that one can efficiently navigate from a node to (i) its parent node, (ii) its first child, (iii) its next sibling, and (iv) its previous sibling (if they exist). For top dags [3], it was shown that a single navigation step can be done in time $O(\log |t|)$. Using the right binary encoding, we can prove the same result for TSLPs: Let r be the maximal rank of a node of the unranked tree t. We define the binary encoding $\mathrm{bin}(t)$ by adding for every node v of rank $s \leq r$ a binary tree of depth $\lceil \log s \rceil$ with s many leaves, whose root is v and whose leaves are the children of v. This introduces at most $2s$ many new binary nodes, which are labelled by a new symbol. We get $|\mathrm{bin}(t)| \leq 3|t|$. In particular, we obtain a TSLP of size $O\big(\frac{n}{\log_\sigma n}\big)$ for $\mathrm{bin}(t)$, where $n = |t|$ and $\sigma = |\mathrm{labels}(t)|$. Note that a traversal step in the initial tree t (going to the parent node, first child, next sibling, or previous sibling) can be simulated by $O(\log r)$ many traversal steps in $\mathrm{bin}(t)$ (going to the parent node, left child, or right child). But for a binary tree s, it was recently shown that a TSLP $\mathcal{G}$ for s can be represented in space $O(|\mathcal{G}|)$ such that a single traversal step takes time $O(1)$ [21] (this generalizes a corresponding result for strings [12]). Hence, we can navigate in t in time $O(\log r) \leq O(\log |t|)$.

5 Arithmetical Circuits

In this section, we present our main application of Theorem 7. Let $\mathcal{S} = (S, +, \cdot)$ be a (not necessarily commutative) semiring. Thus, $(S, +)$ is a commutative monoid with identity element 0, $(S, \cdot)$ is a monoid with identity element 1, and $\cdot$ left and right distributes over $+$. We use the standard notation of arithmetical formulas and circuits over $\mathcal{S}$: An *arithmetical formula* (resp. *arithmetical circuit*) is a binary tree (resp. dag) where internal nodes are labelled with the semiring operations $+$ and $\cdot$, and leaf nodes are labelled with variables $y_1, y_2, \ldots$ or the constants 0 and 1. The *depth* of a circuit is the length of a longest path from the root node to a leaf. An arithmetical circuit evaluates to a multivariate noncommutative polynomial $p(y_1, \ldots, y_n)$ over $\mathcal{S}$, where $y_1, \ldots, y_n$ are the variables occurring at the leaf nodes. Two arithmetical circuits are equivalent if they evaluate to the same polynomial. Brent [4] has shown that every arithmetical formula of size n over a commutative ring can be transformed into an equivalent circuit of depth $O(\log n)$ and size $O(n)$ (the proof easily generalizes to semirings). Using Theorem 7 we can refine the size bound to $O(\frac{n \cdot \log m}{\log n})$, where m is the number of different variables in the formula:

▶ **Theorem 9.** *An arithmetical formula F of size n with m different variables can be transformed in time $O(n \log n)$ into an arithmetical circuit C of depth $O(\log n)$ and size $O(\frac{n \cdot \log m}{\log n})$ such that C and F are equivalent for every semiring.*

Proof sketch. Fix a semiring $\mathcal{S}$. We apply our TSLP construction to the formula tree F and obtain a TSLP $\mathcal{G}$ for F of size $O(\frac{n \cdot \log m}{\log n})$ and depth $O(\log n)$. Using the main construction from [24] we can reduce the rank of nonterminals in $\mathcal{G}$ to 1. Thereby the size and depth of the TSLP only increase by constant factors. Recall that for a nonterminal $A(x)$, $\mathrm{val}_{\mathcal{G}}(A)$ is a tree in which each of the parameter x occurs exactly once. By evaluating this tree in the polynomial semiring $\mathcal{S}[y_1, \ldots, y_m]$, we obtain a noncommutative polynomial $p_A(x) = a_0 + a_1 x a_2$, where $a_0, a_1, a_2 \in \mathcal{S}[y_1, \ldots, y_m]$. We now transform $\mathcal{G}$ into an arithmetical circuit that contains for every nonterminal A of rank one gates that evaluate to the above polynomials a_0, a_1, a_2.

E.g., for a rule of the form $A(x) \to B(C(x))$ one has to substitute the polynomial $p_C(x)$ into $p_B(x)$ and carry out the obvious simplifications. For a nonterminal A of rank 0, the circuit simply contains a gate that evaluates to the polynomial to which $\mathrm{val}_{\mathcal{G}}(A)$ evaluates. ◀

Theorem 9 can also be shown for fields instead of semirings. In this case, the expression is built up using variables, the constants -1, 0, 1, and the field operations $+, \cdot$ and $/$.

6 Future work

In [27] a universal (in the information-theoretic sense) code for binary trees is developed. This code is computed in two phases: In a first step, the minimal dag for the input tree is constructed. Then, a particular binary encoding is applied to the dag. It is shown that the *average redundancy* of the resulting code converges to zero (see [27] for definitions) for every probability distribution on binary trees that satisfies the so-called domination property (a somewhat technical condition) and the representation ratio negligibility property. The latter means that the average size of the dag divided by the tree size converges to zero for the underlying probability distribution. This is, for instance, the case for the uniform distribution, since the average size of the dag is $\Theta(n/\sqrt{\log n})$ [11]. We are confident that replacing the minimal dag by a TSLP of size $O(\frac{n}{\log n})$ in the universal tree encoder from [27] leads to stronger results. In particular, we hope to get a code whose *maximal pointwise redundancy* converges to zero for certain probability distributions. For strings, such a result was obtained in [16] using the fact that every string of length n has an SLP of size $O(\frac{n}{\log n})$.

Another interesting question is whether the time bound of $O(n \log n)$ for the construction of a TSLP of size $O(\frac{n}{\log_\sigma n})$ can be improved to $O(n)$. Related to this is the question for the worst-case output size of the grammar-based tree compressor from [15]. It works in linear time and produces a TSLP that is only by a factor $O(\log n)$ larger than an optimal TSLP. From a complexity theoretic point of view, it would be also interesting to see, whether our TSLP construction can be carried out in logarithmic space or even NC^1.

In [3] the authors proved that the top dag of a given tree t of size n is at most by a factor $\log n$ larger than the minimal dag of t. It is not clear, whether the TSLP constructed by our algorithm has this property too. The construction of the top dag is done in a bottom-up way, and as a consequence identical subtrees are compressed in the same way. This property is crucial for the comparison with the minimal dag. Our algorithm works in a top-down way. Hence, it is not guaranteed that identical subtrees are compressed in the same way.

Finally, one should also study whether all the operations from [3] for top dags can be implemented with the same time bounds also for TSLPs. For traversing the tree, this is possible, see the paragraph at the end of Section 4. For the other operations, like for instance computing lowest common ancestors, this is not clear.

Acknowledgment. We have to thank Anna Gál for helpful comments.

References

1 T. Akutsu. A bisection algorithm for grammar-based compression of ordered trees. *Inf. Process. Lett.*, 110(18-19):815–820, 2010.

2 V. Arvind, S. Raja, and A. V. Sreejith. On lower bounds for multiplicative circuits and linear circuits in noncommutative domains. In *Proc. CSR 2014*, volume 8476 of *LNCS*, pages 65–76. Springer, 2014.

3 P. Bille, I. L. Gørtz, G. M. Landau, and O. Weimann. Tree compression with top trees. In *Proc. ICALP (1) 2013*, volume 7965 of *LNCS*, pages 160–171. Springer, 2013.

4 R. P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, 1974.

5 N. H. Bshouty, R. Cleve, and W. Eberly. Size-depth tradeoffs for algebraic formulas. *SIAM J. Comput.*, 24(4):682–705, 1995.

6 G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Inf. Syst.*, 33(4–5):456–474, 2008.

7 M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005.

8 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`.

9 N. de Bruijn. A combinatorial problem. *Nederl. Akad. Wet., Proc.*, 49:758–764, 1946.

10 P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, 1980.

11 P. Flajolet, P. Sipala, and J.-M. Steyaert. Analytic variations on the common subexpression problem. In *Proc. ICALP 1990*, volume 443 of *LNCS*, pages 220–234. Springer, 1990.

12 L. Gasieniec, R. M. Kolpakov, I. Potapov, and P. Sant. Real-time traversal in grammar-based compressed files. In *Proc. DCC 2005*, page 458. IEEE Computer Society, 2005.

13 M. A. Heap and M. R. Mercer. Least upper bounds on OBDD sizes. *IEEE Trans. Computers*, 43(6):764–767, 1994.

14 D. Hucke, M. Lohrey, and E. Noeth. Constructing small tree grammars and small circuits for formulas. arXiv.org, `http://arxiv.org/abs/1407.4286`, 2014.

15 A. Jèz and M. Lohrey. Approximation of smallest linear tree grammars. In *Proc. STACS 2014*, volume 25 of *LIPIcs*, pages 445–457. Leibniz-Zentrum für Informatik, 2014.

16 J. C. Kieffer and E.-H. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. Inf. Theory*, 46(3):737–754, 2000.

17 J. C. Kieffer, E.-H. Yang, G. J. Nelson, and P. C. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Trans.Inf. Theory*, 46(4):1227–1245, 2000.

18 D. E. Knuth. *The Art of Computer Programming, Vol. I: Fundamental Algorithms*. Addison-Wesley, 1968.

19 P. M. Lewis II, R. E. Stearns, and J. Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *Proc. 6th Annual IEEE Symp. Switching Circuit Theory and Logic Design*, pages 191–202, 1965.

20 H.-T. Liaw and C.-S. Lin. On the obdd-representation of general boolean functions. *IEEE Trans. Computers*, 41(6):661–664, 1992.

21 M. Lohrey. Traversing grammar-compressed trees with constant delay. `http://www.eti.uni-siegen.de/ti/veroeffentlichungen/14-traversal.pdf`.

22 M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

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

24 M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *J. Comput. Syst. Sci.*, 78(5):1651–1669, 2012.

25 W. L. Ruzzo. Tree–size bounded alternation. *J. Comput. Syst. Sci.*, 21:218–235, 1980.

26 P. M. Spira. On time-hardware complexity tradeoffs for boolean functions. In *Proc. 4th Hawaii Symp. on System Sciences*, pages 525–527, 1971.

27 J. Zhang, E.-H. Yang, and J. C. Kieffer. A universal grammar-based code for lossless compression of binary trees. *IEEE Trans. Inf. Theory*, 60(3):1373–1386, 2014.