Traversing Grammar-Compressed Trees with Constant Delay

Traversing Grammar-Compressed Trees with Constant Delay

Markus Lohrey*, Sebastian Maneth†, and Carl Philipp Reh*

*Universität Siegen
Hölderlinstraße 3
57076 Siegen, Germany
{lohrey|reh}@eti.uni-siegen.de

†University of Edinburgh
Crichton Street
Edinburgh, EH8 9AB, UK
smaneth@inf.ed.ac.uk

Abstract

A grammar-compressed ranked tree is represented with a linear space overhead so that a single traversal step, i.e., the move to the parent or the ith child, can be carried out in constant time. The data structure is extended so that equality of subtrees can be checked in constant time.

1 Introduction

Context-free grammars that produce single strings are a widely studied compact string representation known as *straight-line programs* (SLPs). For instance, the string $(ab)^{1024}$ can be represented by the SLP with the rules $A_0 \to ab$ and $A_i \to A_{i-1}A_{i-1}$ for $1 \leq i \leq 10$. In general, an SLP of size n can produce a string of length $2^{\Omega(n)}$. Besides compressors (e.g. LZ78, RePair, or BISECTION, see [1]) generating an SLP from a given string, algorithmic problems on SLPs such as pattern matching and indexing have been investigated thoroughly, see [2] for a survey. Motivated by applications where large tree structures occur, like XML processing, SLPs have been extended to tree straight-line programs (TSLPs), see [3] for a survey and references. A TSLP is a linear context-free tree grammar that produces a single node-labeled ranked ordered tree. TSLPs generalize dags (directed acyclic graphs), which are widely used as compact tree representation. While dags can share repeated subtrees, TSLPs can also share repeated tree patterns (i.e., connected subgraphs). Every n-node tree over a fixed set of node labels can be produced by a TSLP of size $\mathcal{O}(n/\log n)$ (the information-theoretic limit) [2, Thm. 3]. Various algorithmic problems on TSLPs such as XPath querying, evaluating tree automata, and pattern matching have been studied too [2, Section 4].

In this paper we investigate the problem of navigating in a TSLP-represented tree: given a TSLP $\mathcal{G}$ for a tree t, the task is to precompute in time $O(|\mathcal{G}|)$ an $O(|\mathcal{G}|)$-space data structure that allows to move from a node of t in time $O(1)$ to its parent node or to its ith child and to return in time $O(1)$ the node label of the current node. Here the nodes of t are represented in space $O(|\mathcal{G}|)$ in a suitable way. Such a data structure has been developed for string SLPs in [4]; it allows to move from left to right over the string produced by the SLP requiring time $O(1)$ per move. We first extend the data structure from [4] so that the string can be traversed in a two-way fashion, i.e., in each step we can move either to the left or right neighboring position in constant time. This data structure is then used to navigate in a TSLP-represented tree.

TSLPs are typically used for the compression of ranked trees, i.e., trees where the maximal number r of children of a node is bounded by a constant. In many applications, r is indeed bounded (e.g., $r = 2$ for the following two encodings of unranked trees). For unranked trees where r is unbounded, it is more realistic to require that the data structure supports navigation to (*i*) the parent node, (*ii*) the first child, (*iii*) the right sibling, and (*iv*) the left sibling. We can realize these operations using well-known constant-rank encoding of unranked trees with maximal rank r: (1) in the *first-child/next-sibling encoding* $\mathsf{fcns}(t)$, the left (resp. right) child of a node is the first child (resp., right sibling) in t. Here we can support $O(1)$ time navigation for (*ii*)–(*iv*), but the parent move (*i*) requires $O(r)$ time. (2) The *binary encoding* $\mathrm{bin}(t)$ adds for every node v of rank $s \leq r$ a binary tree of

depth $\lceil \log s \rceil$ with s many leaves, the root of which is v and the leaves of which are the children of v. This introduces at most $2s$ many new binary nodes, i.e., $|\mathrm{bin}(t)| \leq 3|t|$. Every navigation step in the original tree can be simulated by $O(\log r)$ many navigation steps in $\mathrm{bin}(t)$.

Our second main result concerns subtree equality checks. This is the problem of checking for two given nodes of a tree whether the subtrees rooted at these two nodes are identical. We extend our data structure for tree navigation such that subtree equality checks can be done in time $O(1)$. The problem of checking equality of subtrees occurs in several different contexts, see for instance [5] for details. Typical applications are common subexpression detection, unification, and non-linear pattern matching. For instance, checking whether the pattern $f(x, f(y, y))$ is matched at a certain tree node needs a constant number of navigation steps and a single subtree equality check.

A full version of this paper with further details can be found in [6].

Related work. The ability to navigate efficiently in a tree is a basic prerequisite for most tree querying procedures. For instance, the DOM representation available in web browsers through JavaScript provides tree navigation primitives (see, e.g., [7]). Tree navigation has been intensively studied in the context of succinct tree representations. Here, the goal is to represent a tree by a bit string, whose length is asymptotically equal to the information-theoretic lower bound. For instance, for binary trees of n nodes the information-theoretic lower bound is $2n + o(n)$ and there exist succinct representations (e.g., the balanced parentheses representation) that encode a binary tree of size n by a bit string of length $2n+o(n)$. In addition there exist such encodings that allow to navigate in the tree in constant time (and support many other tree operations), see e.g. [8] for a survey. Recently, grammatical formalisms for the compression of unranked trees have been proposed as well, such as the top dags of [9]. Top dags can be seen as a variant of TSLPs for unranked trees. Every n-node tree has a top dag of size $O(n \cdot \log\log n / \log n)$ [10]; it is open whether the bound can be improved to the information-theoretic limit $O(n/\log n)$. The navigation problem for top dags is studied in [9]. The authors show that a single navigation step in t can be carried out in time $O(\log |t|)$ in the top dag. Nodes are represented by their preorder numbers, which need $O(\log |t|)$ bits. In [11] an analogous result has been shown for unranked trees that are represented by string SLPs for their balanced parentheses representation. This result covers also TSLPs: From a TSLP $\mathcal{G}$ for a tree t one can compute in linear time an SLP for the balanced bracket representation of t. In some sense our results are orthogonal to the results of [11]: We can navigate, determine node labels, and check equality of subtrees in time $O(1)$, but our representation of tree nodes needs space $O(|\mathcal{G}|)$, whereas Bille et al. [11] can navigate and do some other tree queries in time $O(\log |t|)$, but their node representation (preorder numbers) only needs space $O(\log |t|) \leq O(|\mathcal{G}|)$.

Checking equality of subtrees is trivial for minimal dags, since every subtree is uniquely represented. For so called SL grammar-compressed dags (which can be seen as TSLPs with certain restrictions) it was shown in [12] that equality of subtrees can be checked in time $O(\log |t|)$ for given preorder numbers.

2 Preliminaries

For an alphabet Σ, Σ^* denotes the set of all strings over Σ including the empty string ϵ. For a string $w = a_1 \cdots a_n$ ($a_i \in \Sigma$) we denote by $\mathsf{alph}(w)$ the set of symbols $\{a_1, \ldots, a_n\}$ occurring in w. Moreover, let $|w| = n$, $w[i] = a_i$ and $w[i : j] = a_i \cdots a_j$ where $w[i : j] = \varepsilon$, if $i > j$. Let $w[: i] = w[1 : i]$ and $w[i :] = w[i : n]$.

A *straight-line program (SLP)* is a triple $\mathcal{P} = (N, \Sigma, \mathsf{rhs})$, where N is a finite set of *nonterminals*, Σ is a finite set of *terminals* ($\Sigma \cap N = \emptyset$), and $\mathsf{rhs} : N \to (N \cup \Sigma)^*$ is a mapping such that the binary relation $\{(A, B) \in N \times N \mid B \in \mathsf{alph}(\mathsf{rhs}(A))\}$ is acyclic. This condition ensures that every nonterminal $X \in N$ produces a unique string $\mathsf{val}_{\mathcal{P}}(X) \in \Sigma^*$. It is obtained from the string X by repeatedly replacing nonterminals A by $\mathsf{rhs}(A)$ until no nonterminal occurs. We write $A \to \alpha$ if $\mathsf{rhs}(A) = \alpha$ and call it a *rule* of $\mathcal{P}$. Usually, an SLP has a start nonterminal, but for our purpose it is more convenient to consider SLPs without a start nonterminal. The *size* of $\mathcal{P}$ is

$|\mathcal{P}| = \sum_{A \in N} |\mathsf{rhs}(A)|$, i.e., the total length of all right-hand sides. A simple induction shows that for every SLP $\mathcal{P}$ of size m and every nonterminal A, $|\mathsf{val}_{\mathcal{P}}(A)| \in O(3^{m/3})$ [1, proof of Lemma 1]. On the other hand, it is straightforward to define an SLP $\mathcal{P}$ of size $2n$ such that $|\mathsf{val}(\mathcal{P})| \geq 2^n$. Hence, an SLP can be seen as a compressed string representation that can achieve exponential compression ratios. In Section 5 we need the following algorithmic facts about SLPs (see [2]).

Proposition 1 Let $\mathcal{P}$ and $\mathcal{P}'$ be SLPs and $i, j \in \mathbb{N}$. In polynomial time, one can compute (*i*) the symbol $\mathsf{val}(\mathcal{P})[i]$, (*ii*) an SLP for the string $\mathsf{val}(\mathcal{P})[i:j]$, and (*iii*) the length of the longest common prefix of $\mathsf{val}(\mathcal{P})$ and $\mathsf{val}(\mathcal{P}')$.

We now extend SLPs to trees. A ranked alphabet is a set Σ such that every $a \in \Sigma$ has an associated rank $\mathsf{rank}(a) \in \mathbb{N}$. Let $\Sigma_i = \{a \in \Sigma \mid \mathsf{rank}(a) = i\}$. Fix ranked alphabets $\mathcal{F}$ and $\mathcal{N}$ of *terminal symbols* and *nonterminal symbols* such that for every $i \geq 0$, $\mathcal{F}_i$ and $\mathcal{N}_i$ are countably infinite. Moreover, let $\mathcal{X} = \{x_1, x_2, \dots\}$ be the set of *parameters*. We assume that the three sets $\mathcal{F}$, $\mathcal{N}$, and $\mathcal{X}$ are pairwise disjoint. A *labeled tree* $t = (V, E, \lambda)$ is a finite, directed and ordered tree t with set of nodes V, set of edges $E \subseteq V \times \mathbb{N} \times V$, and labeling function $\lambda : V \to \mathcal{F} \cup \mathcal{N} \cup \mathcal{X}$. We require for every node $v \in V$ that if $\lambda(v) \in \mathcal{F}_k \cup \mathcal{N}_k$, then v has k distinct children $u_1, \dots, u_k$, i.e., $(v, i, u) \in E$ if and only if $1 \leq i \leq k$ and $u = u_i$. A leaf of t is a node with zero children. We require that every node v with $\lambda(v) \in \mathcal{X}$ is a leaf of t. The *size* of t is $|t| = |V|$. We denote trees by their usual term notation, e.g. $a(b, c)$ denotes the tree with an a-labeled root node that has a first child labeled b and a second child labeled c. We define $\mathcal{T}$ as the set of all labeled trees. Let $\mathsf{labels}(t) = \{\lambda(v) \mid v \in V\}$. For $\mathcal{L} \subseteq \mathcal{F} \cup \mathcal{N} \cup \mathcal{X}$ we let $\mathcal{T}(\mathcal{L}) = \{t \in \mathcal{T} \mid \mathsf{labels}(t) \subseteq \mathcal{L}\}$. The tree $t \in \mathcal{T}$ is *linear* if there do not exist different leaves that are labeled with the same parameter. A *tree straight-line program (TSLP)* is a triple $\mathcal{G} = (N, S, \mathsf{rhs})$, where $N \subseteq \mathcal{N}$ is a finite set of *nonterminals*, $S \in \mathcal{N}_0 \cap N$ is the *start nonterminal*, and $\mathsf{rhs} : N \to \mathcal{T}(\mathcal{F} \cup N \cup \mathcal{X})$ is a mapping so that (1) for every $A \in N$, the tree $\mathsf{rhs}(A)$ is linear and if $A \in \mathcal{N}_k$ ($k \geq 0$) then $\mathcal{X} \cap \mathsf{labels}(\mathsf{rhs}(A)) = \{x_1, \dots, x_k\}$, and (2) the binary relation $\{(A, B) \in N \times N \mid B \in \mathsf{labels}(\mathsf{rhs}(A))\}$ is acyclic. These conditions ensure that from every nonterminal $A \in N \cap \mathcal{N}_k$ exactly one linear tree $\mathsf{val}_{\mathcal{G}}(A) \in \mathcal{T}(\mathcal{F} \cup \{x_1, \dots, x_k\})$ is derived by applying the rules $A \to \mathsf{rhs}(A)$ as rewrite rules in the usual sense. More generally, for every tree $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{N} \cup \{x_1, \dots, x_n\})$ we can derive the unique tree $\mathsf{val}_{\mathcal{G}}(t) \in \mathcal{T}(\mathcal{F} \cup \{x_1, \dots, x_n\})$ by applying the rules of $\mathcal{G}$. The tree defined by $\mathcal{G}$ is $\mathsf{val}(\mathcal{G}) = \mathsf{val}_{\mathcal{G}}(S)$. As for SLPs, we also write $A \to t$ if $\mathsf{rhs}(A) = t$. The following example shows the derivation of $\mathsf{val}(\mathcal{G})$ for a TSLP $\mathcal{G}$.

Example 1 Let $\mathcal{G} = (\{S, A, B, \dots, F\}, S, \mathsf{rhs})$, $a \in \mathcal{F}_0$, $b \in \mathcal{F}_2$, and rhs be given by the rules $S \to A(B)$, $A \to C(F, x_1)$, $B \to E(F)$, $C \to D(E(x_1), x_2)$, $D \to b(x_1, x_2)$, $E \to D(F, x_1)$, and $F \to a$. We obtain the derivation $S \to A(B) \to C(F, B) \to D(E(F), B) \to b(E(F), B) \to b(D(F, F), B) \to b(b(F, F), B) \to b(b(a, F), B) \to b(b(a, a), B) \to b(b(a, a), E(F)) \to b(b(a, a), D(F, F)) \to b(b(a, a), b(F, F)) \to b(b(a, a), b(a, F)) \to b(b(a, a), b(a, a)) = \mathsf{val}(\mathcal{G})$.

The *size* $|\mathcal{G}|$ of a TSLP $\mathcal{G} = (N, S, \mathsf{rhs})$ is defined as $|\mathcal{G}| = \sum_{A \in N} |\mathsf{rhs}(A)|$. A TSLP $\mathcal{G} = (N, \mathsf{rhs}, S)$ is *monadic* if $N \subseteq \mathcal{N}_0 \cup \mathcal{N}_1$, i.e., every nonterminal has rank at most one. We make use of the following results.

Proposition 2 (cf. [2, Theorem 1]) From a given TSLP $\mathcal{G}$, where r and k are the maximal ranks of terminal and nonterminal symbols appearing in a right-hand side, one can construct in time $O(r \cdot k \cdot |\mathcal{G}|)$ a monadic TSLP $\mathcal{H}$ such that $\mathsf{val}(\mathcal{H}) = \mathsf{val}(\mathcal{G})$ and $|\mathcal{H}| \in O(r \cdot |\mathcal{G}|)$.

Proposition 3 (cf. [2, Theorem 5]) For TSLPs $\mathcal{G}_1$ and $\mathcal{G}_2$ one can check in polynomial time whether $\mathsf{val}(\mathcal{G}_1) = \mathsf{val}(\mathcal{G}_2)$.

Let us finally explain our computation model. In the following sections, we use the word RAM model, where registers have a certain bit length w. Arithmetic operations and comparisons of registers can be done in time $O(1)$. The space of a data structure is measured by the number of registers. Our algorithms need the following register lengths w, where $\mathcal{G}$ is the input TSLP and $t = \mathsf{val}(\mathcal{G})$.

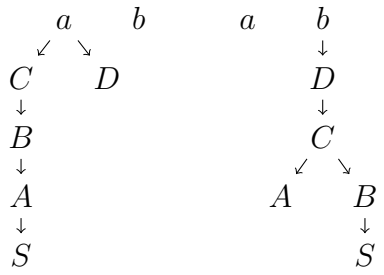

Figure 1: The tries $T_L(a)$, $T_L(b)$ (left), and $T_R(a)$, $T_R(b)$ (right) for the SLP from Example 2.

- For navigation (Section 4) we need a bit length of $w = O(\log |\mathcal{G}|)$, since we only have to store numbers of length at most $|\mathcal{G}|$.
- For equality checks (Section 5) we need a bit length of $w = O(\log |t|) \leq O(|\mathcal{G}|)$, which is the same assumption as in [9, 11].

3 Two-Way Traversal in SLP-Compressed Strings

In [4] the authors present a data structure of size $O(|\mathcal{P}|)$ for storing an SLP $\mathcal{P}$ that allows to produce $\mathsf{val}_{\mathcal{P}}(A)$ with time delay of $O(1)$ per symbol. That is, the symbols of $\mathsf{val}_{\mathcal{P}}(A)$ are produced from left to right and for each symbol constant time is needed. In a first step, we enhance the data structure from [4] for traversing SLP-compressed strings in such a way that both operations of moving to the left and right symbol are supported in constant time. For this, we assume that the SLP $\mathcal{P} = (N, \Sigma, \mathsf{rhs})$ has the property that $|\mathsf{rhs}(A)| = 2$ for each $A \in N$. Every SLP $\mathcal{P}$ with $|\mathsf{val}(\mathcal{P})| \geq 2$ can be transformed in linear time into an SLP $\mathcal{P}'$ with this property and so that $\mathsf{val}(\mathcal{P}') = \mathsf{val}(\mathcal{P})$, see, e.g., [6]. Note that the positions in $\mathsf{val}_{\mathcal{P}}(X)$ correspond to root-leaf paths in the (binary) derivation tree of $\mathcal{P}$ that is rooted in the nonterminal X. We represent such a path by merging successive edges where the path moves in the same direction (left or right) towards the leaf. To formalize this idea, we define for every $\alpha \in N \cup \Sigma$ the strings $L(\alpha), R(\alpha) \in N^*\Sigma$ inductively as follows: for $a \in \Sigma$ let $L(a) = R(a) = a$. For $A \in N$ with $\mathsf{rhs}(A) = \alpha\beta$ ($\alpha, \beta \in N \cup \Sigma$) let $L(A) = AL(\alpha)$ and $R(A) = AR(\beta)$. Note that for every $A \in N$, the string $L(A)$ has the form $A_1A_2 \cdots A_n a$ with $A_i \in N$, $A_1 = A$, and $a \in \Sigma$. We define $\omega_L(A) = a$. The terminal $\omega_R(A)$ is defined analogously by referring to the string $R(A)$.

Example 2 Let $\mathcal{P} = (\{S, A, B, C, D\}, \{a, b\}, \mathsf{rhs})$, where rhs is given by $S \to AB$, $A \to BC$, $B \to CC$, $C \to aD$, $D \to ab$. Then $L(S) = SABCa$, $L(A) = ABCa$, $L(B) = BCa$, $L(C) = Ca$, $L(D) = Da$, $R(S) = SBCDb$, $R(A) = ACDb$, $R(B) = BCDb$, $R(C) = CDb$, $R(D) = Db$. Moreover, $\omega_L(X) = a$ and $\omega_R(X) = b$ for all $X \in \{S, A, B, C, D\}$.

We store all strings $L(A)$ (for $A \in N$) in $|\Sigma|$ many tries: fix $a \in \Sigma$ and let $w_1, \ldots, w_n$ be all strings $L(A)$ such that $\omega_L(A) = a$. Let v_i be the string w_i reversed. Then, $a, v_1, \ldots, v_n$ is a prefix-closed set of strings (except that the empty string is missing) that can be stored in a trie $T_L(a)$. Formally, the nodes of $T_L(a)$ are the strings $a, v_1, \ldots, v_n$, where each node is labeled by its last symbol (so the root is labeled with a), and there is an edge from aw to awA for all appropriate $w \in N^*$ and $A \in N$. The tries $T_R(a)$ are defined in the same way by referring to the strings $R(A)$. Note that the total number of nodes in all tries $T_L(a)$ ($a \in \Sigma$) is exactly $|N| + |\Sigma|$. In fact, every $\alpha \in N \cup \Sigma$ occurs exactly once as a node label in the forest $\{T_L(a) \mid a \in \Sigma\}$. Figure 1 shows the tries $T_L(a)$, $T_L(b)$, $T_R(a)$, and $T_R(b)$ for the SLP from Example 2. Next, we define two alphabets L and R: $L = \{(A, \ell, \alpha) \mid \alpha \in \mathsf{alph}(L(A)) \setminus \{A\}\}$ and $R = \{(A, r, \beta) \mid \beta \in \mathsf{alph}(R(A)) \setminus \{A\}\}$. Note

that the sizes of these alphabets are quadratic in the size of $\mathcal{P}$. On the alphabets L and R we define the partial operations $\mathsf{reduce}_L : L \to L$ and $\mathsf{reduce}_R : R \to R$ as follows. Let $(A, \ell, \alpha) \in L$. We can write $L(A)$ as $Au\alpha v$ for some strings u and v. If $u = \varepsilon$, then $\mathsf{reduce}_L(A, \ell, \alpha)$ is undefined. Otherwise, we can write u as $u'B$ for some $B \in N$. Then we define $\mathsf{reduce}_L(A, \ell, \alpha) = (A, \ell, B)$. The definition of reduce_R is analogous: if $(A, r, \alpha) \in R$, then we can write $R(A)$ as $Au\alpha v$ for some strings u and v. If $u = \varepsilon$, then $\mathsf{reduce}_R(A, r, \alpha)$ is undefined. Otherwise, we can write u as $u'B$ for some $B \in N$ and define $\mathsf{reduce}_R(A, r, \alpha) = (A, r, B)$.

Example 3 (Example 2 continued) The sets L and R are $L = \{(S, \ell, A), (S, \ell, B), (S, \ell, C), (S, \ell, a), (A, \ell, B), (A, \ell, C), (A, \ell, a), (B, \ell, C), (B, \ell, a), (C, \ell, a), (D, \ell, a)\}$ and $R = \{(S, r, B), (S, r, C), (S, r, D), (S, r, b), (A, r, C), (A, r, D), (A, r, b), (B, r, C), (B, r, D), (B, r, b), (C, r, D), (C, r, b), (D, r, b)\}$. For instance, $\mathsf{reduce}_L(S, \ell, a) = (S, \ell, C)$ and $\mathsf{reduce}_R(B, r, D) = (B, r, C)$ whereas $\mathsf{reduce}_L(S, \ell, A)$ is undefined.

An element (A, ℓ, α) can be represented by a pair (v_1, v_2) of different nodes in the forest $\{T_L(a) \mid a \in \Sigma\}$, where v_1 (resp. v_2) is the unique node labeled with α (resp., A). Note that v_1 and v_2 belong to the same trie and that v_2 is below v_1. This observation allows us to reduce the computation of the mapping reduce_L to a so-called *next link query*: from the pair (v_1, v_2) we have to compute the unique child v of v_1 such that v is on the path from v_1 to v_2. If v is labeled with B, then $\mathsf{reduce}_L(A, \ell, \alpha) = (A, \ell, B)$, which is represented by the pair (v, v_2). Clearly, the same remark applies to the map reduce_R. The following result is mentioned in [4], see Section 6 for a discussion.

Proposition 4 A trie T can be represented in space $O(|T|)$ such that any next link query can be answered in time $O(1)$. Moreover, this representation can be computed in time $O(|T|)$ from T.

We represent a path in the derivation tree of $\mathcal{P}$ with root X by a sequence of triples

$$\gamma = (A_1, d_1, A_2)(A_2, d_2, A_3) \cdots (A_{n-1}, d_{n-1}, A_n)(A_n, d_n, a) \in (L \cup R)^+$$

such that $n \geq 1$, $A_1 = X$, $a \in \Sigma$, and, for $1 \leq i \leq n-1$, $d_i = \ell$ if and only if $d_{i+1} = r$. We call such a sequence a *valid X-sequence for $\mathcal{P}$* in the following, or briefly a valid sequence if X is not important and $\mathcal{P}$ is clear from the context. Note that γ indeed defines a unique path in the derivation tree rooted at X that ends in an a-labeled leaf. This path, in turn, defines in the string $\mathsf{val}_{\mathcal{P}}(X)$ a unique position that we denote by $\mathsf{pos}(\gamma)$.

One can now define a procedure right (and analogously a procedure left) that transforms a valid X-sequence γ into a valid X-sequence γ' such that $\mathsf{pos}(\gamma') = \mathsf{pos}(\gamma)+1$ in case the latter is defined (and otherwise returns "undefined"), see [6] for details. It is based on the obvious fact that in order to move in a full binary tree from a leaf to the next leaf (where "next" refers to the natural left-to-right order on the leaves) one has to repeatedly move to parent nodes as long as right-child edges are traversed (in the opposite direction); when this is no longer possible, the current node is the left child of its parent p. One now moves to the right child of p and from here repeatedly to left children until a leaf is reached. Each of these four operations can be implemented in constant time on valid sequences, using the fact that (*i*) consecutive edges to left (resp., right) children are merged into a single triple from L (resp., R) in our representation of paths and (*ii*) that the mappings reduce_L and reduce_R can be computed in constant time by Proposition 4. Thereby γ is used as a stack that is only modified at its right end.

4 Traversal in TSLP-Compressed Trees

In this section, we extend the traversal algorithm from the previous section from SLPs to TSLPs. We only consider monadic TSLPs. If the TSLP is not monadic, then we can transform it into a monadic TSLP using Proposition 2. We fix the monadic TSLP $\mathcal{G} = (N, \mathsf{rhs}, S)$. One can modify $\mathcal{G}$ so that for all $A \in N$, $\mathsf{rhs}(A)$ has one of the following four forms (we write x for the parameter x_1), see [6] for details:

(a) $B(C)$ for $B, C \in N$ (b) $B(C(x))$ for $B, C \in N$ (c) $a \in \mathcal{F}_0$

(d) $f(A_1, \ldots, A_{i-1}, x, A_{i+1}, \ldots, A_n)$ for $A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n \in N$, $f \in \mathcal{F}_n$, $n \geq 1$.

We write N_x ($x \in \{a, b, c, d\}$) for the set of all nonterminals whose right-hand side is of the above type (x). Let $N_1 = N_a \cup N_b$ and $N_2 = N_c \cup N_d$. Note that if we start with a nonterminal $A \in N_a$ and then replace nonterminals from N_1 by their right-hand sides repeatedly, we obtain a tree that consists of nonterminals from N_d followed by a single nonterminal from N_c. After replacing these nonterminals by their right-hand sides, we obtain a caterpillar tree which is composed of right-hand sides of the form (d) followed by a single constant from $\mathcal{F}_0$. Hence, there is a unique path of terminal symbols from $\mathcal{F}$, and we call this path the *spine path* of A. All other nodes of the caterpillar tree are leaves and labeled with nonterminals of rank zero to which we can apply again the TSLP rules. The size of a caterpillar tree and therefore a spine path can be exponential in the size of the TSLP.

Given the TSLP of the above form, we define *its derived SLP* $\mathcal{P} = (N_1, N_2, \mathsf{rhs}_1)$. If $A \in N_1$ with $\mathsf{rhs}(A) = B(C)$ or $\mathsf{rhs}(A) = B(C(x))$, then $\mathsf{rhs}_1(A) = BC$. The triple alphabets L and R defined above Example 3 refer to this SLP $\mathcal{P}$. Moreover, we define M to be the set of all triples (A, k, A_k) where $A \in N_d$, $\mathsf{rhs}(A) = f(A_1, \ldots, A_{i-1}, x, A_{i+1}, \ldots, A_n)$ and $k \in \{1, \ldots, n\} \setminus \{i\}$.

Note that the nodes of the tree $\mathsf{val}(\mathcal{G})$ can be identified with the nodes of $\mathcal{G}$'s derivation tree that are labeled with a nonterminal from N_2 (every nonterminal from N_2 has a unique occurrence of a terminal symbol on its right-hand side). A *valid sequence* for $\mathcal{G}$ is a sequence

$$\gamma = (A_1, e_1, A_2)(A_2, e_2, A_3) \cdots (A_{n-1}, e_{n-1}, A_n)(A_n, e_n, A_{n+1}) \in (L \cup R \cup M)^*$$

such that $n \geq 0$, $e_1, \ldots, e_n \in \{\ell, r\} \uplus \mathbb{N}$, if $S \in N_a$ then $n \geq 1$, if $n \geq 1$ then $A_1 = S$ and $A_{n+1} \in N_2$, and if $e_i, e_{i+1} \in \{\ell, r\}$ then $e_i = \ell$ if and only if $e_{i+1} = r$. Such a valid sequence represents a path in the derivation of the TSLP $\mathcal{G}$ from the root to an N_2-labeled node, and hence represents a node of the tree $\mathsf{val}(\mathcal{G})$. Note that in case $S \in N_c$ the empty sequence is valid too and represents the root of the single-node tree $\mathsf{val}(\mathcal{G})$. Moreover, if the last triple (A_n, e_n, A_{n+1}) belongs to M, then we must have $A_{n+1} \in N_c$, i.e., $\mathsf{rhs}(A_{n+1}) \in \mathcal{F}_0$. Here is an example.

Example 4 Consider the monadic TSLP $\mathcal{G}$ with nonterminals $S, A, B, \ldots, I$ and rules $S \to A(B)$, $A \to C(D(x))$, $B \to C(E)$, $C \to f(F, x)$ $D \to f(x, F)$, $E \to D(F)$, $F \to G(H)$, $G \to I(I(x))$, $H \to a$, and $I \to g(x)$. We have $N_1 = \{S, A, B, E, F, G\}$ and $N_2 = \{C, D, I\}$. The SLP $\mathcal{P}$ consists of the rules $S \to AB$, $A \to CD$, $B \to CE$, $E \to DF$, $F \to GH$, and $G \to II$ (the terminal symbols are C, D, H, I). The triple set M is $M = \{(C, 1, F), (D, 2, F)\}$. A valid sequence is for instance $(S, \ell, A)(A, r, D)(D, 2, F)(F, \ell, I)$.

Using valid sequences of $\mathcal{G}$ and the SLP-traversal algorithms sketched in the previous section, it is straightforward to do a single navigation step in constant time. Let us fix a valid sequence γ. We consider the following possible navigation steps: move to the parent node (if it exists) and move to the ith child (if it exists). Consider for instance the navigation to the ith child (similar arguments can be used to navigate to the parent node). If γ is empty or ends with a triple from $M \cup (N \times \{\ell, r\} \times N_c)$, then γ represents a leaf node of $\mathsf{val}(\mathcal{G})$; hence the ith child does not exist. Otherwise let $\beta \neq \varepsilon$ be the maximal suffix of γ, which belongs to $(L \cup R)^*$. Then β represents a path in the derivation tree of the string SLP $\mathcal{P}$ that is rooted in a certain nonterminal $A \in N_a$ and that leads to a certain nonterminal $B \in N_d$. This path corresponds to a node of $\mathsf{val}_{\mathcal{G}}(A)$ that is located on the spine path of A. We can now apply our SLP-navigation algorithms left and right to the sequence β in order to move up or down on the spine path. More precisely, let β end with (C, d, B) ($d \in \{\ell, r\}$, $B \in N_d$). We can now distinguish the following cases, where $f(A_1, \ldots, A_{j-1}, x, A_{j+1}, \ldots, A_n)$ is the right-hand side of B: (*i*) if $i \neq j$, then we obtain the ith child by appending to γ the triple $(B, i, A_i) \in M$ followed by the path that represents the root of A_i (which consists of at most one triple). (*ii*) If $i = j$, then we obtain the ith child of the current node by moving down on the spine path. Thus, we replace the suffix β by $\mathsf{right}(\beta)$. The following theorem summarizes the above discussion.

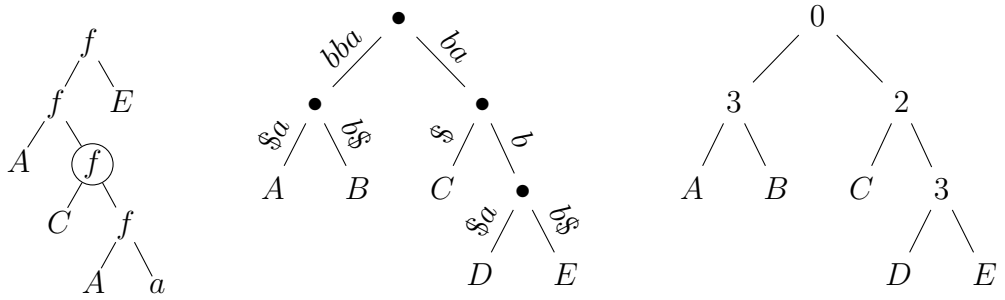

Figure 2: Caterpillar tree (left), Patricia tree (middle) and its modified version (right)

Theorem 1 Given a monadic TSLP $\mathcal{G}$ we can compute in linear time on a word RAM with register length $O(\log|\mathcal{G}|)$ a data structure of size $O(|\mathcal{G}|)$ that allows to do the following computations in time $O(1)$, where γ is a valid sequence that represents the tree node v: (*i*) compute the valid sequence for the parent node of v, and (*ii*) compute the valid sequence for the ith child of v.

5 Equality checks

Consider a monadic TSLP $\mathcal{G} = (N, \mathsf{rhs}, S)$, where again the right-hand side of every rule has one of the four forms (a)–(d) shown at the beginning of Section 4. Let $t = \mathsf{val}(\mathcal{G})$. The goal of this section is to extend the navigation algorithm from the previous section such that for two nodes of t (represented by valid sequences) we can test in $O(1)$ time whether the subtrees rooted at the two nodes are equal. The rough idea is as follows. Recall from the beginning of Section 4 the string SLP $\mathcal{P}$ derived from the TSLP $\mathcal{G}$. We show that subtree equality checks for the tree t can be reduced to queries of the following form: is the length of the longest common suffix of $\mathsf{val}_{\mathcal{P}}(A)[:n_A]$ and $\mathsf{val}_{\mathcal{P}}(B)[:n_B]$ at least k? Here $A, B \in N_a$ and the n_X ($X \in N_a$) are precomputed positions. Using a Patricia tree data structure, these queries can be answered in constant time.

Recall from the end of Section 2 that we use the word RAM model with registers of length $O(\log|t|)$ in this section. The same assumption is also used in [9, 11]. As before, the preprocessed data structure has size $O(|\mathcal{G}|)$ and the query time is $O(1)$. However this time, the preprocessing time is polynomial in the TSLP size $|\mathcal{G}|$ and not just linear. It will be hard to reduce this to linear time preprocessing. The best known algorithm for checking equality of SLP-compressed strings has quadratic complexity [13], and from two SLPs $\mathcal{P}_1$ and $\mathcal{P}_2$ we can easily compute a TSLP for a tree t whose root has two children in which $\mathsf{val}(\mathcal{P}_1)$ and $\mathsf{val}(\mathcal{P}_2)$ are rooted as linear chains.

We assume that $\mathcal{G}$ is *reduced* in the sense that $\mathsf{val}_{\mathcal{G}}(A) \neq \mathsf{val}_{\mathcal{G}}(B)$ for all $A, B \in N$ with $A \neq B$. Reducing the TSLP does not increase its size and the reduction can be done in polynomial time (recall that we allow polynomial time preprocessing) by Proposition 3. To motivate the forthcoming definitions, we first give an example of two equal subtrees produced by a single TSLP.

Example 5 Consider the reduced TSLP $\mathcal{G}$ with the rules $S \to G(A)$, $A \to a$, $B \to f(A,x)$ $C \to B(A)$, $D \to f(C,x)$, $E \to D(C)$, $F \to f(x,E)$, $G \to H(I(x))$, $H \to F(B(x))$, $I \to D(B(x))$. The caterpillar tree of S is given in the left of Figure 2. The subtree rooted in the node that is marked by a circle is the same as the one produced by E.

We use the notations introduced in the previous section. For a string $w = A_1A_2\cdots A_nA_{n+1} \in N_d^*N_c$ we define $\mathsf{val}_{\mathcal{G}}(w) = \mathsf{val}_{\mathcal{G}}(A_1(A_2(\cdots A_n(A_{n+1})\cdots)))$. The SLP $\mathcal{P} = (N_1, N_2, \mathsf{rhs}_1)$ derived from $\mathcal{G}$ as defined in the beginning of Section 4 is: if $A \in N_1$ with $\mathsf{rhs}(A) = B(C)$ or $\mathsf{rhs}(A) = B(C(x))$, then $\mathsf{rhs}_1(A) = BC$. So, for every $A \in N_a$ we have $\mathsf{val}_{\mathcal{P}}(A) =$

$A_1A_2\cdots A_nA_{n+1}$ for some $n \geq 1$, where $A_i \in N_d$ for $1 \leq i \leq n$ and $A_{n+1} \in N_c$. Let $\ell(A) = n+1$ (this is the length of the spine path of A), $A[i:j] = A_iA_{i+1}\cdots A_j$, $A[i:] = A_i\cdots A_nA_{n+1}$, and $A[:i] = A_1\cdots A_i$. We define $s(A)$ as the smallest number $i \geq 2$ such that $\mathsf{val}_{\mathcal{G}}(A[i:]) = \mathsf{val}_{\mathcal{G}}(B)$ for some nonterminal $B \in N$ of rank zero. This unique nonterminal B is denoted by A'. Moreover, let $r_A(x) = \mathsf{rhs}(A_{s(A)-1})$ be the right-hand side of $A_{s(A)-1} \in N_d$. Hence, $r_A(x)$ is a tree of the form $f(X_1,\ldots,X_{i-1},x,X_{i+1},\ldots,X_m)$ for $X_1,\ldots,X_{i-1},X_{i+1},\ldots,X_m \in N$, $f \in \mathcal{F}_m$, $m \geq 1$. With these notations, we have $\mathsf{val}_{\mathcal{G}}(A[s(A)-1:]) = \mathsf{val}_{\mathcal{G}}(r_A(A'))$. Note that $s(A)$, r_A, and A' are well-defined since $\mathsf{val}_{\mathcal{G}}(A[n+1:]) = \mathsf{val}_{\mathcal{G}}(A_{n+1})$ and A_{n+1} has rank zero.

Example 5 (Continued) The derived SLP $\mathcal{P}$ of $\mathcal{G}$ has the rules $S \to GA$, $C \to BA$, $E \to DC$, $G \to HI$, $H \to FB$, $I \to DB$. We have $\mathsf{val}_{\mathcal{P}}(S) = FBDBA$, $\mathsf{val}_{\mathcal{G}}(DBA) = f(f(a,a),f(a,a)) = \mathsf{val}_{\mathcal{G}}(E)$, but $\mathsf{val}_{\mathcal{G}}(BDBA) = f(a,f(f(a,a),f(a,a)))$ is not equal to one of the trees $\mathsf{val}_{\mathcal{G}}(X)$ for a nonterminal X of rank zero. Hence, we have $S' = E$, $s(S) = 3$, and $r_S = \mathsf{rhs}(B) = f(A,x)$.

The proofs of the following two lemmas are presented in the long version [6] of this paper.

Lemma 1 For every nonterminal $A \in N_a$, we can compute $s(A), r_A$ and A' in polynomial time.

Lemma 2 For $A, B \in N_a$ and $1 \leq i < s(A)$, $1 \leq j < s(B)$, the following two conditions are equivalent: (*i*) $\mathsf{val}_{\mathcal{G}}(A[i:]) = \mathsf{val}_{\mathcal{G}}(B[j:])$ and (*ii*) $\mathsf{val}_{\mathcal{P}}(A[i : s(A)-2]) = \mathsf{val}_{\mathcal{P}}(B[j : s(B)-2])$ and $r_A(A') = r_B(B')$.

Consider a valid sequence $\gamma \in (L \cup R \cup M)^*$ for $\mathcal{G}$. We can uniquely factorize γ as

$$\gamma = \gamma_1(A_1,k_1,B_1)\gamma_2(A_2,k_2,B_2)\cdots\gamma_{n-1}(A_{n-1},k_{n-1},B_{n-1})\gamma_n, \tag{1}$$

where $\gamma_i \in (L \cup R)^*$ $(1 \leq i \leq n)$ and $(A_i,k_i,B_i) \in M$ $(1 \leq i \leq n-1)$. To simplify the notation, let us set $B_0 = S$. Hence, every γ_i is either empty or a valid B_{i-1}-sequence for the SLP $\mathcal{P}$, and we have defined the position $\mathsf{pos}(\gamma_i)$ in the string $\mathsf{val}_{\mathcal{P}}(B_{i-1})$ according to Section 3. It is easy to modify our traversal algorithms from the previous section such that for every $1 \leq i \leq n$ we store in the sequence γ also the nonterminal B_{i-1} and the number $\mathsf{pos}(\gamma_i)$ right after γ_i (if $\gamma_i \neq \varepsilon$), i.e., just before (A_i,k_i,B_i). We do not explicitly write these nonterminals and positions in valid sequences in order to not complicate the notation.

We would like to use Lemma 2 for equality checks. To do this, we have to assume that $\mathsf{pos}(\gamma_i) < s(B_{i-1})$ in (1) for every $1 \leq i \leq n$ with $\gamma_i \neq \varepsilon$ (this corresponds to the assumptions $1 \leq i < s(A)$ and $1 \leq j < s(B)$ in Lemma 2). To do this, we have to modify the traversal algorithms sketched in the previous section as follows. Assume that the current valid sequence is γ from (1). As remarked above, we store the numbers $\mathsf{pos}(\gamma_i)$ right after each γ_i. Assume that the final number $\mathsf{pos}(\gamma_n)$ has reached the value $s(B_{n-1})-1$ and we want to move to the ith child of the current node. We proceed as described at the end of Section 4 with one exception: in Case (*ii*) (before Theorem 1) we would increase $\mathsf{pos}(\gamma_i)$ to $s(B_{n-1})$. To avoid this, we start a new valid sequence for the root of the tree $\mathsf{val}_{\mathcal{G}}(B'_{n-1})$. Note that by the definition of B'_{n-1}, this is exactly the tree rooted at the ith child of the node represented by γ. So, we can continue the traversal in the tree $\mathsf{val}_{\mathcal{G}}(B'_{n-1})$. Therefore, we continue with the sequence $\gamma \mid \mathsf{root}(B'_{n-1})$, where $\mid$ is a separator symbol, and $\mathsf{root}(B'_{n-1})$ is the sequence that represents the root of $\mathsf{val}_{\mathcal{G}}(B'_{n-1})$ (either ε or a triple of the form (B'_{n-1},ℓ,C) for some $C \in N_d$). The navigation to the parent node can be easily adapted as well.

Let us now consider two sequences γ_1 and γ_2 (that may contain the separator symbol $\mid$ as explained in the previous paragraph). Let v_i be the node of $\mathsf{val}(\mathcal{G})$ represented by γ_i and let t_i be the subtree of $\mathsf{val}(\mathcal{G})$ rooted in v_i. We want to check in time $O(1)$ whether $t_1 = t_2$. We can first compute in time $O(1)$ the labels of the nodes v_1 and v_2, respectively. In case one of these labels belongs to $\mathcal{F}_0$ (i.e., one of the nodes v_1, v_2 is a leaf) we can easily determine whether $t_1 = t_2$. Hence, we can assume that neither v_1 nor v_2 is a leaf. In particular we can assume that $\gamma_1 \neq \varepsilon \neq \gamma_2$

(recall that ε is a valid sequence only in case $\mathsf{val}(\mathcal{G})$ consists of a single node) and that neither γ_1 nor γ_2 ends with a triple from M. Let us factorize γ_i as $\gamma_i = \alpha_i\beta_i$, where β_i is the maximal suffix of γ_i that belongs to $(L \cup R)^*$. Hence, we have $\beta_1 \neq \varepsilon \neq \beta_2$.

Assume that β_i is a valid C_i-sequence of $\mathcal{P}$, and that $n_i = \mathsf{pos}(\beta_i)$. Thus, the suffix β_i represents the n_ith leaf of the derivation tree of $\mathcal{P}$ with root C_i. Recall that we store C_i and n_i at the end of the sequence γ_i. Hence, we have constant time access to C_i and n_i. We have $C_i \in N_a$ and $n_i < s(C_i)$. With the notation introduced before, we get $t_i = \mathsf{val}_{\mathcal{G}}(C_i[n_i :])$. Since $n_1 < s(C_1)$ and $n_2 < s(C_2)$, Lemma 2 implies that $t_1 = t_2$ if and only if the following two conditions hold: (*i*) $\mathsf{val}_{\mathcal{P}}(C_1[n_1 : s(C_1) - 2]) = \mathsf{val}_{\mathcal{P}}(C_2[n_2 : s(C_2) - 2])$ and (*ii*) $r_{C_1}(C_1') = r_{C_2}(C_2')$.

Condition (*ii*) can be checked in time $O(1)$, since we can precompute in polynomial time r_A and A' for every $A \in N_a$ by Lemma 1. So, let us concentrate on Condition (*i*). First, we check whether $s(C_1) - n_1 = s(C_2) - n_2$. If not, then the lengths of $\mathsf{val}_{\mathcal{P}}(C_1[n_1 : s(C_1) - 2])$ and $\mathsf{val}_{\mathcal{P}}(C_2[n_2 : s(C_2) - 2])$ differ and we cannot have equality. Hence, assume that $k := s(C_1) - 1 - n_1 = s(C_2) - 1 - n_2$. Let ℓ be the length of the longest common suffix of $\mathsf{val}_{\mathcal{P}}(C_1[: s(C_1) - 2])$ and $\mathsf{val}_{\mathcal{P}}(C_2[: s(C_2) - 2])$. Then, it remains to check whether $k \leq \ell$. Clearly, in space $O(|\mathcal{G}|)$ we cannot store explicitly all these lengths ℓ for all $C_1, C_2 \in N_a$. Instead, we precompute in polynomial time a modified Patricia tree for the set of strings $w_A := \mathsf{val}_{\mathcal{P}}(A[: s(A) - 2])^{\mathrm{rev}}\$$ ($\$$ is a new symbol that is appended in order make the set of strings prefix-free and w^{rev} is the string w reversed) for $A \in N_a$. Then, we need to compute in time $O(1)$ the length of the longest common prefix for two of these strings w_A and w_B. Recall that the Patricia tree for a set of strings $w_1, \ldots, w_n$ is obtained from the trie for the prefixes of the w_i by eliminating nodes with a single child. But instead of labeling edges of the Patricia tree with factors of the w_i, we label every internal node with the length of the strings that lead from the root to the node. Let us give an example instead of a formal definition.

Example 6 Consider the strings $w_A = abba\$$, $w_B = abbb\$$, $w_C = ba\$$, $w_D = baba\$$ and $w_E = babb\$$. Figure 2 shows their Patricia tree (center) and the modified Patricia tree (right).

Since our modified Patricia tree has $|N_a|$ many leaves (one for each $A \in N_a$) and every internal node has at least two children, we have at most $2|N_a| - 1$ many nodes in the tree and every internal node is labeled with a $(\log |t|)$-bit number (note that the length of every string $\mathsf{val}_{\mathcal{P}}(A)$ ($A \in N_a$) is bounded by $|t|$. Hence, on the word RAM model we can store the modified Patricia tree in space $O(|\mathcal{G}|)$. Finally, the length of the longest common prefix of two string w_A and w_B can be obtained by computing the lowest common ancestor of the two leaves corresponding to the strings w_A and w_B in the Patricia tree. The number stored in the lowest common ancestor is the length of the longest common prefix of w_A and w_B. Using a data structure for computing lowest common ancestors in time $O(1)$ [14, 15] we obtain an $O(1)$-time implementation of subtree equality checking. Finally, from Proposition 1 it follows that the modified Patricia tree for the strings w_A ($A \in N_a$) can be precomputed in polynomial time. The following theorem is the main result of this section.

Theorem 2 Given a monadic TSLP $\mathcal{G}$ for a tree $t = \mathsf{val}(\mathcal{G})$ we can compute in polynomial time on a word RAM with register length $O(\log |t|)$ a data structure of size $O(|\mathcal{G}|)$ that allows to do the following computations in time $O(1)$, where γ and γ' are valid sequences (as modified in this section) that represent the tree nodes v and v', respectively: (*i*) compute the valid sequence for the parent node of v, (*ii*) compute the valid sequence for the ith child of v, and (*iii*) check whether the subtrees rooted in v and v' are equal.

6 Discussion

We have presented a data structure to traverse grammar-compressed ranked trees with constant delay and to check equality of subtrees. The solution is based on the ideas of [4] and the fact that next link queries can be answered in time $O(1)$ (after linear time preprocessing). It would be interesting to develop an efficient implementation of the technique. Next link queries can be implemented in

many different ways. The solution given in [4] is based on a variant of the lowest common ancestor (LCA) algorithm due to Schieber and Vishkin [15] (described in [16]). Another alternative is to store the first-child/next-sibling encoded binary trees of the tries $T_L(a)$ and $T_R(a)$ for $a \in \Sigma$. The first-child/next-sibling encoding is defined for ordered trees, whereas the tries $T_L(a)$ and $T_R(a)$ are unordered. Hence we order the children of a node in an arbitrary way. Then the next link of v_1 above v_2 is equal to the LCA of v_2 and the last child of v_1 in the original trie. This observation allows to use simple and efficient LCA data structures like the one from [14].

[1] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat, "The smallest grammar problem," *IEEE Trans. Inf. Theory*, vol. 51, pp. 2554–2576, 2005.

[2] M. Lohrey, "Algorithmics on SLP-compressed strings: A survey," *Groups Complexity Cryptology*, vol. 4, pp. 241–299, 2012.

[3] ——, "Grammar-based tree compression," in *Proc. DLT 2015*, 2015, pp. 46–57.

[4] L. Gasieniec, R. M. Kolpakov, I. Potapov, and P. Sant, "Real-time traversal in grammar-based compressed files," in *Proc. DCC 2005*, 2005, p. 458.

[5] J. Cai and R. Paige, "Using multiset discrimination to solve language processing problems without hashing," *Theor. Comput. Sci.*, vol. 145, pp. 189–228, 1995.

[6] M. Lohrey, S. Maneth, and C. P. Reh, "Traversing grammar-compressed trees with constant delay," arXiv.org, Tech. Rep., 2015, http://arxiv.org/abs/1511.02141.

[7] O. Delpratt, R. Raman, and N. Rahman, "Engineering succinct DOM," in *Proc. EDBT 2008*, 2008, pp. 49–60.

[8] G. Navarro and K. Sadakane, "Fully functional static and dynamic succinct trees," *ACM Trans. Algorithms*, vol. 10, pp. 16:1–16:39, 2014.

[9] P. Bille, I. L. Gørtz, G. M. Landau, and O. Weimann, "Tree compression with top trees," *Inf. Comput.*, vol. 243, pp. 166–177, 2015.

[10] L. Hübschle-Schneider and R. Raman, "Tree compression with top trees revisited," in *Proc. SEA 2015*, 2015, pp. 15–27.

[11] P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. R. Satti, and O. Weimann, "Random access to grammar-compressed strings and trees," *SIAM J. Comput.*, vol. 44, pp. 513–539, 2015.

[12] M. Bousquet-Mélou, M. Lohrey, S. Maneth, and E. Noeth, "XML compression via DAGs," *Theor. Comput. Syst.*, vol. 57, no. 4, pp. 1322–1371, 2015.

[13] A. Jez, "Faster fully compressed pattern matching by recompression," *ACM Transactions on Algorithms*, vol. 11, pp. 20:1–20:43, 2015.

[14] M. A. Bender and M. Farach-Colton, "The LCA problem revisited," in *Proc. LATIN 2000*, 2000, pp. 88–94.

[15] B. Schieber and U. Vishkin, "On finding lowest common ancestors: Simplification and parallelization," *SIAM J. Comput.*, vol. 17, pp. 1253–1262, 1988.

[16] D. Gusfield, *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.