Traversing Grammar-Compressed Trees with Constant Delay

Markus Lohrey[1] and Sebastian Maneth[s]

[1] Universität Siegen, Germany
[2] University of Edinburgh, 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 i-th child, can be carried out in constant time.

1 Introduction

Context-free grammars that produce single strings are a widely studied compact string representation and are also known as *straight-line programs* (SLPs). For instance, the string $(ab)^{1024}$ 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). In general, an SLP of size n can produce a string of length $2^{\Omega(n)}$. Besides grammar-based compressors (e.g. LZ78, RePair, or BISECTION, see [5] for more details) that derive an SLP from a given string, also algorithmic problems on SLP-compressed strings such as pattern matching, indexing, and compressed word problems have been investigated thoroughly, see [12] for a survey.

Motivated by applications where large tree structures occur, like XML processing, SLPs have been extended to node-labelled ranked trees [3, 4, 9, 11, 14]. In those papers, straight-line linear context-free tree grammars are used. Such grammars produce a single tree and are also known as tree straight-line programs (TSLPs). TSLPs generalize dags (directed acyclic graphs), which are widely used as compact tree representation. Whereas dags only allow to share repeated subtrees, TSLPs can also share repeated internal tree patterns (i.e., connected subgraphs). The grammar-based tree compressor from [9] produces for every tree (for a fixed set of node labels) a TSLP of size $\mathcal{O}(\frac{n}{\log n})$, which is worst-case optimal. Various querying problems on TSLP-compressed trees such as XPath querying and evaluating tree automata are studied in [13, 15, 17].

In this paper we study the problem of navigating in a TSLP-compressed tree: Given a TSLP $\mathcal{G}$ for a tree t, the task is to construct in linear time an $O(|\mathcal{G}|)$-space data structure that allows to move from a node of t in constant time to its parent node or to its i-th child. Here the nodes of t should be represented in space $O(|\mathcal{G}|)$ in a suitable way. Such a data structure has been developed for string SLPs in [8]; 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 [8] 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 traverse the paths of a TSLP.

The ability to navigate efficiently in a tree is a basic prerequisite for most tree querying procedures. For instance, the DOM representation available in all 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 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. [18] for a survey.

Our algorithm only works for monadic TSLPs, i.e., TSPS where every nonterminal has at most one parameter. This is not a severe restriction: By a result from [15], every TSLP of size n can be transformed into a monadic TSLP of size $O(r \cdot n)$, where r is the maximal rank (number of children) of a node in the generated tree. In many applications, r is 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 by using a suitable constant-rank encoding of trees. Two folklore binary tree encodings of an unranked tree t with maximal rank of a node r, are:

1. First-child/next-sibling encoding $\mathsf{fcns}(t)$: The left (resp. right) child of a node in $\mathsf{fcns}(t)$ is the first child (resp., right sibling) in t. On this encoding, we can support $O(1)$ time navigation for (*ii*)–(*iv*) of above. The parent move (*i*) however, requires $O(r)$ time.
2. Binary encoding: 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 (labelled by a new symbol). Thus $|\mathrm{bin}(t)| \leq 3|t|$. Every navigation step in the original tree can be simulated by $O(\log r)$ many navigation steps in $\mathrm{bin}(t)$.

Note that from a TSLP for t one can easily construct a TSLP for the encoded trees ($\mathsf{fcns}(t)$ or $\mathsf{bin}(t)$).

In [2] the authors consider so called top dags as a compact tree representation. Top dag can be seen as a slight variant of TSLPs for an unranked tree. It is shown in [2] that for every tree of size n the top dag has size $O(\frac{n}{\log^{0.19} n})$. Moreover, also the navigation problem for top dags is studied in [2]. The authors show that a single navigation step in t can be done in time $O(\log |t|)$ in the top dag. By the above remark, we obtain the same time bound if we take a TSLP for the binary encoding $\mathsf{bin}(t)$. In addition, we obtain, at least in theory, the better size bound of $O(\frac{n}{\log n})$ for TSLPs, using the compressor of [9]. But it remains to compare the top dags with TSLPs on real world data.

An implementation of navigation over TSLP-compressed trees is given in [16]. Their worst-case time per navigation step is $O(h)$ where h is the height of the TSLP. The authors demonstrate that on XML data full traversals are 5–7 times slower than over succinct trees (based on an implementation by Sadakane) while using 3–15 times less space.

2 Preliminaries

For an alphabet Σ we denote by Σ^* 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.

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

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

Tree Straight-Line Programs. For every $i \geq 0$, we fix a countable infinite set $\mathcal{F}_i$ of *terminals* of rank i and a countable infinite set $\mathcal{N}_i$ of *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 $\mathcal{X} = \{x_1, x_2, \ldots\}$ be a countable infinite set of *parameters*. We assume that the three sets $\mathcal{F}$, $\mathcal{N}$, and $\mathcal{X}$ are pairwise disjoint. A *labelled 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 labelling 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 $u_1, \ldots, u_k$, 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 $|t| = |\{v \in V \mid \lambda(v) \in \mathcal{F} \cup \mathcal{N}\}|$, i.e., we do not count parameters. We denote trees in their usual term notation, e.g. $a(b, c)$ denotes the tree with an a-labelled root node that has a first child labelled b and a second child labeled c. We define $\mathcal{T}$ as the set of all labelled trees. Let $\mathrm{labels}(t) = \{\lambda(v) \mid v \in V\}$. For $\mathcal{L} \subseteq \mathcal{F} \cup \mathcal{N} \cup \mathcal{X}$ we let $\mathcal{T}(\mathcal{L}) = \{t \in \mathcal{T} \mid \mathrm{labels}(t) \subseteq \mathcal{L}\}$. The tree $t \in \mathcal{T}$ is *linear* if there do not exist different leaves that are labelled with the same parameter.

We now define a particular form of context-free tree grammars (see [6] 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 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 such that the following hold:

- For every $A \in N$, the tree $\mathsf{rhs}(A)$ is linear and if $A \in \mathcal{N}_k$ ($k \geq 0$) then $\mathcal{X} \cap \mathrm{labels}(\mathsf{rhs}(A)) = \{x_1, \ldots, x_k\}$.
- The binary relation $\{(A, B) \in N \times N \mid B \in \mathrm{labels}(\mathsf{rhs}(A))\}$ is acyclic.

These conditions ensure that from every nonterminal $A \in N \cap \mathcal{N}_k$ exactly one linear tree $\mathsf{val}_{\mathcal{G}}(A) \in \mathcal{T}(\mathcal{F} \cup \{x_1, \ldots, x_k\})$ is derived by using the rules $A \to \mathsf{rhs}(A)$ as rewrite rules in the usual sense. The tree defined by $\mathcal{G}$ is $\mathsf{val}(\mathcal{G}) = \mathsf{val}_{\mathcal{G}}(S)$. Instead of giving a formal definition, we show a derivation of $\mathsf{val}(\mathcal{G})$ from S in an example:

Example 1. Let $\mathcal{G} = (\{S, A, B, C, D, E, F\}, S, \mathsf{rhs})$, $a \in \mathcal{F}_0$, and $b \in \mathcal{F}_2$, where

$$\mathsf{rhs}(S) = A(B),\ \mathsf{rhs}(A) = C(F, x_1),\ \mathsf{rhs}(B) = E(F),\ \mathsf{rhs}(C) = D(E(x_1), x_2),$$
$$\mathsf{rhs}(D) = b(x_1, x_2),\ \mathsf{rhs}(E) = D(F, x_1),\ \mathsf{rhs}(F) = a.$$

A possible derivation of $\mathsf{val}(\mathcal{G}) = b(b(a, a), b(a, a))$ from S is:

$$\begin{aligned} S &\to A(B) \to C(F, B) \to D(E(F), B) \to b(E(F), B) \to b(D(F, F), B) \\ &\to b(b(F, F), B) \to b(b(a, F), B) \to b(b(a, a), B) \to b(b(a, a), E(F)) \\ &\to b(b(a, a), D(F, F)) \to b(b(a, a), b(F, F)) \to b(b(a, a), b(a, F)) \\ &\to b(b(a, a), b(a, a)). \end{aligned}$$

The size $|\mathcal{G}|$ of a TSLP $\mathcal{G} = (N, S, \mathsf{rhs})$ is defined as $|\mathcal{G}| = \sum_{A \in N} |\mathsf{rhs}(A)|$ (recall that we do not count parameters). For instance, the TSLP from Example 1 has size 12.

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 1. The following result is shown in [15].

Proposition 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* $|\mathcal{H}| \in O(r \cdot |\mathcal{G}|)$.

3 Two-Way Traversal in SLP-Compressed Strings

In [8] the authors presented a data structure of size $O(|\mathcal{H}|)$ for storing an SLP $\mathcal{H}$ that allows to produce $\mathsf{val}_{\mathcal{H}}(A)$ with time delay of $O(1)$ per symbol. That is, the symbols of $\mathsf{val}_{\mathcal{H}}(A)$ are produced from left to right and for each symbol constant time is needed.

In a first step, we enhance the data structure from [8] 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 $\mathcal{H} = (N, \Sigma, \mathsf{rhs})$ has the property that $|\mathsf{rhs}(A)| = 2$ for each $A \in N$. Every SLP $\mathcal{H}$ with $|\mathsf{val}(\mathcal{H})| \geq 2$ can easily be transformed in polynomial time into an SLP $\mathcal{H}'$ with this property and $\mathsf{val}(\mathcal{H}') = \mathsf{val}(\mathcal{H})$: First, we replace all occurrences of nonterminals B with $|\mathsf{rhs}(B)| \leq 2$ by $\mathsf{rhs}(B)$. If $\mathsf{rhs}(S) = A \in N$, then we redefine $\mathsf{rhs}(S) = \mathsf{rhs}(A)$. Finally, for every $A \in N$ such that $\mathsf{rhs}(A) = \alpha_1 \cdots \alpha_n$ with $n \geq 3$ and $\alpha_1, \ldots, \alpha_n \in (N \cup \Sigma)$ we introduce new nonterminals $A_2, \ldots, A_{n-1}$ with rules $A_i \to \alpha_i A_{i+1}$ for $2 \leq i \leq n-2$, and $A_{n-1} \to \alpha_{n-1}\alpha_n$, and redefine $\mathsf{rhs}(A) = \alpha_1 A_2$. It should be clear that $|\mathcal{H}'| \leq 2 \cdot |\mathcal{H}|$.

Note that the positions in $\mathsf{val}_{\mathcal{H}}(X)$ correspond to root-leaf paths in the (binary) derivation tree of $\mathcal{H}$ 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

$$L(a) = R(a) = a.$$

$$\begin{tikzcd}
 & a \arrow[dl] \arrow[dr] & b & & a & b \arrow[d] & \\
C \arrow[d] & & D & & & D \arrow[d] & \\
B \arrow[d] & & & & & C \arrow[dl] \arrow[dr] & \\
A \arrow[d] & & & & A & & B \arrow[d] \\
S & & & & & & S
\end{tikzcd}$$

Fig. 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 $A \in N$ with $\mathsf{rhs}(A) = \alpha\beta$ ($\alpha, \beta \in N \cup \Sigma$) let

$$L(A) = A\,L(\alpha) \text{ and } R(A) = A\,R(\beta). \tag{1}$$

Note that for every $A \in N$, the string $L(A)$ has the form $A_1 A_2 \cdots A_n a$ 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 $\mathcal{H} = (\{S, A, B, C, D\}, \{a, b\}, \mathsf{rhs})$, where

$$\mathsf{rhs}(S) = AB,\ \mathsf{rhs}(A) = BC,\ \mathsf{rhs}(B) = CC,\ \mathsf{rhs}(C) = aD,\ \mathsf{rhs}(D) = ab.$$

Then we have $L(S) = SABCa$, $L(A) = ABCa$, $L(B) = BCa$, $L(C) = Ca$, $L(D) = Da$, $R(S) = SBCDb$, $R(A) = ACDb$, $R(B) = BCDb$, $R(C) = CDb$, $R(D) = Db$. Moreover, $\omega_L(X) = a$ and $\omega_R(X) = b$ for all $X \in \{S, A, B, C, D\}$.

We can store all strings $L(A)$ (for $A \in N$) in $|\Sigma|$ many tries: For each $a \in \Sigma$ 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 word 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 labelled by its last symbol (so the root is labelled 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 $\{T_L(a) \mid a \in \Sigma\}$.

Example 3 (Example 2 continued). The tries $T_L(a)$, $T_L(b)$, $T_R(a)$, and $T_R(b)$ for the SLP from Example 2 are shown in Figure 1.

Next, we define two alphabets L and R as follows:

$$L = \{(A, \ell, \alpha) \mid \alpha \in \mathsf{alph}(L(A)) \setminus \{A\}\} \tag{2}$$
$$R = \{(A, r, \beta) \mid \beta \in \mathsf{alph}(R(A)) \setminus \{A\}\}. \tag{3}$$

Note that the sizes of these alphabets are quadratic in the size of $\mathcal{H}$. 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 $(A, \ell, \alpha) \in L$. Hence 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 4 (Example 2 continued). The sets L and R are:

$$\begin{aligned} L = \{&(S,\ell,A),(S,\ell,B),(S,\ell,C),(S,\ell,a),(A,\ell,B),\\ &(A,\ell,C),(A,\ell,a),(B,\ell,C),(B,\ell,a),(C,\ell,a),(D,\ell,a)\}\\ R = \{&(S,r,B),(S,r,C),(S,r,D),(S,r,b),(A,r,C),(A,r,D),\\ &(A,r,b),(B,r,C),(B,r,D),(B,r,b),(C,r,D),(C,r,b),(D,r,b)\}. \end{aligned}$$

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 labelled 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 labelled 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 [8], see Section 5 for a discussion.

Proposition 2. *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 $\mathcal{H}$ 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 $n \geq 1$ and the following properties hold:

- $A_1 = X$, $a \in \Sigma$
- $d_i = \ell$ if and only if $d_{i+1} = r$ for all $1 \leq i \leq n-1$.

We call such a sequence a *valid X-sequence for $\mathcal{H}$* in the following, or briefly a valid sequence if X is not important and $\mathcal{H}$ is clear from the context. Note that a valid X-sequence γ indeed defines a unique path in the derivation tree rooted at X that ends in a leaf that is labelled with the terminal symbol a if γ ends with (A, d, a). This path, in turn, defines a unique position in the string $\mathsf{val}_{\mathcal{H}}(X)$ that we denote by $\mathsf{pos}(\gamma)$.

We now define a procedure right (see Algorithm 1) that takes as input a valid X-sequence γ and returns a valid X-sequence γ' such that $\mathsf{pos}(\gamma') = \mathsf{pos}(\gamma) + 1$ in case the latter is defined (and otherwise returns "undefined"). The idea uses the obvious fact that in order to move in a binary tree from a leaf to the next leaf (where "next" refers

Algorithm 1: right(γ)

```
Data: valid sequence $\gamma$
$(A, d, a) :=$ pop($\gamma$) ;
if $d = \ell$ then
    expand-right($\gamma, A, a$)
else
    if $\gamma = \varepsilon$ then
        return undefined
    else
        $(A', \ell, A) :=$ pop($\gamma$) ;
        $\gamma :=$ expand-right($\gamma, A', A$)
    end
end
return $\gamma$
```

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, using the fact that consecutive edges to left (resp., right) children are merged into a single triple from L (resp., R) in our representation of paths. We use the valid sequence γ as a stack with the operations pop (which returns the right-most triple of γ, which is thereby removed from γ) and push (which appends a given triple on the right end of γ). The procedure right uses the procedure expand $-$ right (see Algorithm 2) that takes as input a (non-valid) sequence γ of triples, $A \in N$, and a symbol $\alpha \in N \cup \Sigma$ such that $\gamma(A, \ell, \alpha)$ is a prefix of a valid sequence. The sequence γ has to be treated as a global variable in order to obtain an $O(1)$-time implementation (to make the presentation clearer, we pass γ as a parameter to expand $-$ right).

In a completely analogous way we can define a procedure left that takes as input a valid X-sequence γ and returns a valid X-sequence γ' such that $\mathsf{pos}(\gamma') = \mathsf{pos}(\gamma) - 1$ in case the latter is defined (otherwise "undefined" will be returned). The details are left to the reader.

4 Traversal in TSLP-Compressed Trees

In this section, we extend the traversal algorithm from the previous section from SLPs to TSLPs. We will restrict to monadic TSLPs. If the TSLP is not monadic, then we can transform it into a monadic TSLP using Proposition 1.

Let us fix a *monadic* TSLP $\mathcal{G} = (N, \mathsf{rhs}, S)$. One can easily 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):

(a) $B(C)$ for $B, C \in N$ (and A has rank 0)
(b) $B(C(x))$ for $B, C \in N$ (and A has rank 1)
(c) $f(A_1, \ldots, A_n)$ for $A_1, \ldots, A_n \in N$ (and A has rank 0)

Algorithm 2: expand-right(γ, A, α)

```
Data: sequence $\gamma$, $A \in N$, $\alpha \in N \cup \Sigma$ such that $\gamma(A, \ell, \alpha)$ is a prefix of a valid sequence
let $\mathsf{rhs}(A) = \alpha_1\alpha_2$;
if $\alpha \neq \alpha_1$ then
    $(A, \ell, B) := \mathsf{reduce}_L(A, \ell, \alpha)$ ;
    push($\gamma, (A, \ell, B)$) ;
    let $\mathsf{rhs}(B) = \beta_1\beta_2$;
    push($\gamma, (B, r, \beta_2)$);
    if $\beta_2 \in N$ then
        push($\gamma, (\beta_2, \ell, \omega_L(\beta_2))$)
    end
else
    if $\gamma = \varepsilon$ then
        push($\gamma, (A, r, \alpha_2)$)
    else
        $(B, r, A) :=$ pop($\gamma$) ;
        push($\gamma, (B, r, \alpha_2)$)
    end
    if $\alpha_2 \in N$ then
        push($\gamma, (\alpha_2, \ell, \omega_L(\alpha_2))$)
    end
end
return $\gamma$
```

(d) $f(A_1, \ldots, A_{i-1}, x, A_{i+1}, \ldots, A_n)$ for $A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n \in N$ (and A has rank 1)

This modification is done in a similar way as the transformation of SLPs at the beginning of Section 3: Remove nonterminals $A \in \mathcal{N}_1$ with $\mathsf{rhs}(A) = x_1$ (resp., $\mathsf{rhs}(A) = B(x_1)$, $B \in \mathcal{N}$) by replacing in all right-hand sides every subtree $A(t)$ by t (resp., $B(t)$) and iterate this as long as possible. Similarly, we can eliminate nonterminals $A \in \mathcal{N}_0 \setminus \{S\}$ with $\mathsf{rhs}(A) \in \mathcal{N}$. If $\mathsf{rhs}(S) = B \in \mathcal{N}_0$ then redefine $\mathsf{rhs}(S) := \mathsf{rhs}(B)$. Finally, right-hand sides of size at least two are split top-down into new nonterminals with right-hand sides of the above types. In case the right-hand side contains the parameter x_1 (in a unique position), we decompose along the path to x_1. It should be clear that the resulting TSLP has size $< 2|\mathcal{G}|$.

Let N_1 (resp., N_2) be the set of all nonterminals such that the corresponding right-hand side has the form (a) or (b) (resp., (c) or (d)). We define M to be the set of all triples (A, k, A_k) where $A \in N_2$ and $k \neq i$ in case $\mathsf{rhs}(A)$ has the form (d).

Note that the nodes of the tree $\mathsf{val}(\mathcal{G})$ can be identified with the nodes of the derivation tree that are labelled with a nonterminal from N_2 (every nonterminal from N_2 has a unique occurrence of a terminal symbol on its right-hand side).

We define an SLP $\mathcal{H} = (N_1, N_2, \mathsf{rhs}_1)$ as follows: 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 from (2) and (3) refer to this SLP $\mathcal{H}$. A *valid sequence* for $\mathcal{G}$ is a sequence

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

Algorithm 3: parent(γ)

```
Data: valid sequence $\gamma$
if $\gamma = \varepsilon$ then
    return (undefined)
end
if $\gamma \in (L \cup R)^+$ then
    return left($\gamma$) ;                    (We can have left($\gamma$) = undefined.)
end
if $\gamma \in (L \cup R \cup M)^* \cdot M$ then
    let $\gamma = \alpha \cdot (A, k, B)$ ;
    return $\alpha$ ;                          (Note that $\alpha$ is again valid.)
end
if $\gamma \in (L \cup R \cup M)^* \cdot M \cdot (L \cup R)^+$ then
    let $\gamma = \alpha \cdot (A, k, B) \cdot \beta$ with $\beta \in (L \cup R)^+$ and $(A, k, B) \in M$;
    $\gamma'$ := left($\beta$) ;
    if $\gamma'$ = undefined then
        return $\alpha$ ;                      (Note that $\alpha$ is again valid.)
    else
        return $\alpha \cdot (A, k, B) \cdot \gamma'$
    end
end
```

(note that $e_1, \ldots, e_n \in \{\ell, r\} \uplus \mathbb{N}$) such that $n \geq 0$ and the following hold:

– If $S \in N_1$ then $n \geq 1$.
– If $n \geq 1$ then $A_1 = S$ and $A_{n+1} \in N_2$.
– 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-labelled node, and hence represents a node of the tree $\mathsf{val}(\mathcal{G})$. Note that in case $S \in N_2$ the empty sequence is valid too and represents the root of the tree $\mathsf{val}(\mathcal{G})$. Here is an example:

Example 5. Consider the monadic TSLP $\mathcal{G}$ with nonterminals S, A, B, C, D, E, F, G, H, J and the following productions

$$\begin{aligned}&\mathsf{rhs}(S) = A(B),\ \mathsf{rhs}(A) = C(D(x)),\ \mathsf{rhs}(B) = C(E),\ \mathsf{rhs}(C) = f(F, x),\\ &\mathsf{rhs}(D) = f(x, F),\ \mathsf{rhs}(E) = f(F, F),\ \mathsf{rhs}(F) = G(H),\ \mathsf{rhs}(G) = J(J(x)),\\ &\mathsf{rhs}(H) = a,\ \mathsf{rhs}(J) = g(x).\end{aligned}$$

It produces the tree shown in Figure 2. We have $N_1 = \{S, A, B, F, G\}$ and $N_2 = \{C, D, E, J\}$. The SLP $\mathcal{H}$ consists of the rules

$$S \to AB,\ A \to CD,\ B \to CE,\ F \to GH,\ G \to JJ$$

(the terminal symbols are C, D, E, J). The triple set M is

$$M = \{(C, 1, F), (D, 2, F), (E, 1, F)(E, 2, F)\}.$$

Algorithm 4: child(γ, i)

```
Data: valid sequence $\gamma$, number $i$
if $\gamma = \varepsilon$ then
    let $\mathsf{rhs}(S) = f(A_1, \ldots, A_n)$ with $A_1, \ldots, A_n \in \mathcal{N}_0$;
    if $1 \le i \le n$ then
        return $(S, i, A_i) \cdot \mathrm{root}(A_i)$
    else
        return undefined
    end
end
if $\gamma \in (L \cup R \cup M)^* \cdot (L \cup R)$ then
    let $\gamma = \alpha \cdot \beta$ such that $\beta \in (L \cup R)^+$ and $\alpha \notin (L \cup R \cup M)^* \cdot (L \cup R)$;
    let $\beta$ end with the triple $(A, d, B)$;                    (We must have $B \in N_2$.)
    let $\mathsf{rhs}(B) = f(a_1, \ldots, a_n)$ with $a_j \in \mathcal{N}_0 \cup \{x_1\}$;
    if $1 \le i \le n$ then
        if $a_i = x_1$ then
            return $\alpha \cdot \mathrm{right}(\beta)$
        else
            return $\alpha \cdot \beta \cdot (B, i, a_i) \cdot \mathrm{root}(a_i)$
        end
    else
        return undefined
    end
end
if $\gamma \in (L \cup R \cup M)^* \cdot M$ then
    let $\gamma = \beta \cdot (A, k, B)$;                           (We must have $B \in N_2$.)
    let $\mathsf{rhs}(B) = f(A_1, \ldots, A_n)$ with $A_j \in \mathcal{N}_0$;
    if $1 \le i \le n$ then
        return $\gamma \cdot (B, i, A_i) \cdot \mathrm{root}(A_i)$
    else
        return undefined
    end
end
```

A valid sequence is for instance $(S, \ell, A)(A, r, D)(D, 2, F)(F, \ell, J)$. It represents the circled g-labelled node in Figure 2.

Using valid sequences of $\mathcal{G}$ it is easy 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 i-th child (if it exists). Algorithm 3 shows the pseudo code for moving to the parent node, and Algorithm 4 shows the pseudo code for moving to the i-th child. If this node does not exist, then "undefined" is returned. To make the code more readable we denote the concatenation operator for sequences of triples (or sets of triple sequences) with "$\cdot$". We make use of the procedures left and right from Section 3, applied to the SLP $\mathcal{H}$ derived from our TSLP $\mathcal{G}$. Note that in Algorithm 3 we apply (in case $\gamma \in (L \cup R \cup M)^* \cdot M \cdot (L \cup R)^+$) the procedure left to the maximal suffix β of the current valid sequence γ that consists of triples from $L \cup R$ (and

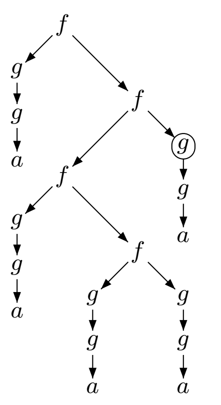

Fig. 2. The tree produced from the TSLP in Example 5

similarly for Algorithm 4). To provide an $O(1)$ time implementation we do not copy the sequence β and pass it to left but apply left directly to γ. The right-most triple from M works as a left-end marker. Algorithm 4 uses the procedure $\mathsf{root}(A)$ (with A of rank 0). This procedure is not shown explicitly: It simply returns ϵ if $A \in N_2$, and otherwise returns $(A, \ell, \omega_L(A))$ (recall ω_L from Page 5). Hence, it returns the representation of the root node of $\mathsf{val}_{\mathcal{G}}(A)$.

5 Discussion

We have presented a data structure to traverse grammar-compressed ranked trees with constant delay. The solution is based on the ideas of [8] and the fact that next link queries can be answered in constant time (after linear time preprocessing). It would be interesting to develop an efficient implementation of the technique. Link queries can be implemented in many different ways. One solution given in [8] is based on a variant of the LCA algorithm due to Schieber and Vishkin [19] (described in [10]). Another solution is to use *level-ancestor queries* (together with depth-queries), as are available in implementation of succinct tree data structures (e.g., the one of Navarro and Sadakane [18]). Another alternative is to store the first-child/next-sibling encoded binary tree of the original tries. The first-child/next-sibling encoding is defined for ordered trees. In our situation, we have to answer next-link queries for the tries $T_L(a)$ and $T_R(a)$ for $a \in \Sigma$, which 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 *lowest common ancestor* of v_2 and the last child of v_1 in the original tree. This observation allows to use simple and efficient lowest common ancestor data structures like the one of Bender and Farach-Colton [1].

It will be interesting to investigate how more complex tree queries than edge traversal, such as e.g. is-descendant, level-ancestor, or lowest-common ancestor queries, can be supported over grammar-compressed trees.

References

1. M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proceedings of LATIN 2000*, LNCS 1776, pages 88–94. Springer, 2000.
2. P. Bille, I. L. Gørtz, G. M. Landau, and O. Weimann. Tree compression with top trees. In *Proceedings of ICALP (1) 2013*, LNCS 7965, pages 160–171. Springer, 2013.
3. M. Bousquet-Mélou, M. Lohrey, S. Maneth, and E. Noeth. XML compression via DAGs. *Theory of Computing Systems*, 2014.
4. G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Information Systems*, 33(4–5):456–474, 2008.
5. M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.
6. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available at `http://tata.gforge.inria.fr/`, 2007.
7. O. Delpratt, R. Raman, and N. Rahman. Engineering succinct DOM. In *Proceedings of EDBT*, pages 49–60, ACM, 2008.
8. L. Gasieniec, R. M. Kolpakov, I. Potapov, and P. Sant. Real-time traversal in grammar-based compressed files. In *Proceedings of DCC 2005*, page 458, IEEE Computer Society, 2005. Availabe at `http://www.csc.liv.ac.uk/~leszek/papers/dcc05.ps.gz`.
9. D. Hucke, M. Lohrey, and E. Noeth. Constructing small tree grammars and small circuits for formulas. In *Proceedings of FSTTCS 2014*, volume 29 of *LIPIcs*, pages 457–468. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.
10. D. Gusfield. Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology. Cambridge University Press, 1997.
11. A. Jeż and M. Lohrey. Approximation of smallest linear tree grammars. In *Proceedings of STACS 2014*, volume 25 of *LIPIcs*, pages 445–457. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.
12. M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.
13. M. Lohrey and S. Maneth. The complexity of tree automata and XPath on grammar-compressed trees. *Theoretical Computer Science*, 363(2):196–210, 2006.
14. M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using RePair. *Information Systems*, 38(8):1150–1167, 2013.
15. M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *Journal of Computer and Systems Science*, 78(5):1651–1669, 2012.
16. S. Maneth and T. Sebastian. Fast and tiny structural self-indexes for XML. *CoRR*, abs/1012.5696, 2010.
17. S. Maneth and T. Sebastian. XPath node selection over grammar-compressed trees. In *Proceedings of TTATT 2013*, Electronic Proceedings in Theoretical Computer Science 134, pages 38–48, 2013.
18. G. Navarro and K. Sadakane. Fully functional static and dynamic succinct trees. *ACM Transactions on Algorithms*, 10(3):16, 2014.
19. B. Schieber and U. Vishkin. On Finding Lowest Common Ancestors: Simplification and Parallelization. *SIAM Journal on Computing*, 17(6):1253–1262, 1988.