Grammar-based Compression of Unranked Trees

Adrià Gascón[1], Markus Lohrey[2], Sebastian Maneth[3], Carl Philipp Reh[2], and Kurt Sieber[2]

[1] Warwick University and Alan Turing Institute, UK
[2] Universität Siegen, Germany
[3] Universität Bremen, Germany

Abstract. We introduce forest straight-line programs (FSLPs) as a compressed representation of unranked ordered node-labelled trees. FSLPs are based on the operations of forest algebra and generalize tree straight-line programs. We compare the succinctness of FSLPs with two other compression schemes for unranked trees: top dags and tree straight-line programs of first-child/next sibling encodings. Efficient translations between these formalisms are provided. Finally, we show that equality of unranked trees in the setting where certain symbols are associative or commutative can be tested in polynomial time. This generalizes previous results for testing isomorphism of compressed unordered ranked trees.

1 Introduction

Generally speaking, grammar-based compression represents an object succinctly by means of a small context-free grammar. In many grammar-based compression formalisms such a grammar can be exponentially smaller than the object. Henceforth, there is a great interest in problems that can be solved in polynomial time on the grammar, while requiring at least linear time on the original uncompressed object. One of the most well-known and fundamental such problems is testing equality of the strings produced by two context-free string grammars, each producing exactly one string (such grammars are also known as straight-line programs — in this paper we use the term string straight-line program, SSLP for short). Polynomial time solutions to this problem were discovered, in different contexts by different groups of people, see the survey [14] for references.

Grammar-based compression has been generalized from strings to ordered ranked node-labelled trees, by means of linear context-free tree grammars generating exactly one tree [6]. Such grammars are also known as tree straight-line programs, TSLPs for short. Equality of the trees produced by two TSLPs can also be checked in polynomial time: one constructs SSLPs for the pre-order traversals of the trees, and then applies the above mentioned result for SSLPs, see [6]. The tree case becomes more complex when *unordered* ranked trees are considered. Such trees can be represented using TSLPs, by simply ignoring the order of children in the produced tree. Checking isomorphism of unordered ranked trees generated by TSLPs was recently shown to be solvable in polynomial time [16]. The solution transforms the TSLPs so that they generate canonical representations of the original trees and then checks equality of these canonical forms.

The aforementioned result for ranked trees cannot be applied to *unranked* trees (where the number of children of a node is not bounded), which arise for instance in

XML document trees. This is unfortunate, because (*i*) grammar-based compression is particularly effective for XML document trees (see [15]), and (*ii*) XML document trees can often be considered unordered (one speaks of "data-centric XML", see e.g. [1, 3, 5, 20]), allowing even stronger grammar-based compressions [17].

In this paper we introduce a generalization of TSLPs and SSLPs that allows to produce ordered unranked node-labelled trees and forests (i.e., ordered sequences of trees) that we call *forest straight-line programs*, FSLPs for short. In contrast to TSLPs, FSLPs can compress very wide and flat trees. For instance, the tree $f(a, a, \ldots, a)$ with n many a's is not compressible with TSLPs but can be produced by an FSLP of size $O(\log n)$. FSLPs are based on the operations of horizontal and vertical forest composition from forest algebras [4]. The main contributions of this paper are the following:

Comparison with other formalisms. We compare the succinctness of FSLPs with two other grammar-based formalisms for compressing unranked node-labelled ordered trees: TSLPs for 'first-child/next-sibling" (fcns) encodings and top dags. The fcns-encoding is the standard way of transforming an unranked tree into a binary tree. Then the resulting binary tree can be succinctly represented by a TSLP. This approach was used to apply the TreeRePair-compressor from [15] to unranked trees. We prove that FSLPs and TSLPs for fcns-encodings are equally succinct up to constant multiplicative factors and that one can change between both representations in linear time (Propositions 9 and 10).

Top dags are another formalism for compressing unranked trees [2]. Top dags use horizontal and vertical merge operations for tree construction, which are very similar to the horizontal and vertical concatenation operations from FSLPs. Whereas a top dag can be transformed in linear time into an equivalent FSLP with a constant multiplicative blow-up (Proposition 6), the reverse transformation (from an FSLP to a top dag) needs time $O(\sigma \cdot n)$ and involves a multiplicative blow-up of size $O(\sigma)$ where σ is the number of node labels of the tree (Proposition 7). A simple example (Example 8) shows that this σ-factor is unavoidable. The reason for the σ-factor is a technical restriction in the definition of top dags: In contrast to FSLPs, top dags only allow sharing of common subtrees but not of common subforests. Hence, sharing between (large) subtrees which only differ in their root labels may be impossible at all (as illustrated by Example 8), and this leads to the σ-blow-up in comparison to FSLPs. The impossibility of sharing subforests would also complicate the technical details of our main algorithmic results for FSLPs (in particular Proposition 10 and Theorem 13 which is discussed below) for which we make heavy use of a particular normal form for FSLPs that exploits the sharing of proper subforests. We therefore believe that at least for our purposes, FSLPs are a more adequate formalism than top dags.

Testing equality modulo associativity and commutativity. Our main algorithmic result for FSLPs can be formulated as follows: Fix a set Σ of node labels and take a subset $\mathcal{C} \subseteq \Sigma$ of "commutative" node labels and a subset $\mathcal{A} \subseteq \Sigma$ of "associative" node labels. This means that for all $a \in \mathcal{A}$, $c \in \mathcal{C}$ and all trees $t_1, t_2, \ldots, t_n$ (*i*) we do not distinguish between the trees $c(t_1, \ldots, t_n)$ and $c(t_{\sigma(1)}, \ldots, t_{\sigma(n)})$, where σ is any permutation (commutativity), and (*ii*) we do not distinguish the trees $a(t_1, \ldots, t_n)$ and $a(t_1, \ldots, t_{i-1}, a(t_i, \ldots, t_{j-1}), t_j, \ldots, t_n)$ for $1 \leq i \leq j \leq n+1$ (associativity). We then show that for two given FSLPs F_1 and F_2 that produce trees t_1 and t_2 (of

possible exponential size), one can check in polynomial time whether t_1 and t_2 are equal modulo commutativity and associativity (Theorem 13). Note that unordered tree isomorphism corresponds to the case $\mathcal{C} = \Sigma$ and $\mathcal{A} = \emptyset$ (in particular we generalize the result from [16] for ranked unordered trees). Theorem 13 also holds if the trees t_1 and t_2 are given by top dags or TSLPs for the fcns-encodings, since these formalisms can be transformed efficiently into FSLPs. Theorem 13 also shows the utility of FSLPs even if one is only interested in say binary trees, which are represented by TSLPs. The law of associativity will yield very wide and flat trees that are no longer compressible with TSLPs but are still compressible with FSLPs.

Missing proofs can be found in the arXiv version of this paper [11].

2 Straight-line programs over algebras

We will produce strings, trees and forests by algebraic expressions over certain algebras. These expressions will be compressed by directed acyclic graphs. In this section, we introduce the general framework, which will be reused several times in this paper.

An algebraic structure is a tuple $\mathcal{A} = (A, f_1, \ldots, f_k)$ where A is the universe and every $f_i \colon A^{n_i} \to A$ is an operation of a certain arity n_i. In this paper, the arity of all operations will be at most two. If $n_i = 0$, then f_i is called a constant. Moreover, it will be convenient to allow partial operations for the f_i. Algebraic expressions over $\mathcal{A}$ are defined in the usual way: if $e_1, \ldots, e_{n_i}$ are algebraic expressions over $\mathcal{A}$, then also $f_i(e_1, \ldots, e_{n_i})$ is an algebraic expressions over $\mathcal{A}$. For an algebraic expression e, $[\![e]\!] \in A$ denotes the element to which e evaluates (it can be undefined).

A *straight-line program* (SLP for short) over $\mathcal{A}$ is a tuple $P = (V, S, \rho)$, where V is a set of *variables*, $S \in V$ is the *start variable*, and ρ maps every variable $A \in V$ to an expression of the form $f_i(A_1, \ldots, A_{n_i})$ (the so called *right-hand side* of A) such that $A_1, \ldots, A_{n_i} \in V$ and the edge relation $E(P) = \{(A, B) \in V \times V \mid B \text{ occurs in } \rho(A)\}$ is acyclic. This allows to define for every variable $A \in V$ its value $[\![A]\!]_P$ inductively by $[\![A]\!]_P = f_i([\![A_1]\!]_P, \ldots, [\![A_{n_i}]\!]_P)$ if $\rho(A) = f_i(A_1, \ldots, A_{n_i})$. Since the f_i can be partially defined, the value of a variable can be undefined. The SLP P will be called *valid* if all values $[\![A]\!]_P$ ($A \in V$) are defined. In our concrete setting, validity of an SLP can be tested by a simple syntax check. The value of P is $[\![P]\!] = [\![S]\!]_P$. Usually, we prove properties of SLPs by induction along the partial order $E(P)^*$.

It will be convenient to allow for the right-hand sides $\rho(A)$ algebraic expressions over $\mathcal{A}$, where the variables from V can appear as atomic expressions. By introducing additional variables, we can transform such an SLP into an equivalent SLP of the original form. We define the size $|P|$ of an SLP P as the total number of occurrences of operations $f_1, \ldots, f_k$ in all right-hand sides (which is the number of variables if all right-hand sides have the standard form $f_i(A_1, \ldots, A_{n_i})$).

Sometimes it is useful to view an SLP $P = (V, S, \rho)$ as a directed acyclic graph (dag) $(V, E(P))$, together with the distinguished output node S, and the node labelling that associates the label f_i with the node $A \in V$ if $\rho(A) = f_i(A_1, \ldots, A_{n_i})$. Note that the outgoing edges $(A, A_1), \ldots, (A, A_{n_i})$ have to be ordered since f_i is in general not commutative and that multi-edges have to be allowed. Such dags are also known as algebraic circuits in the literature.

String straight-line programs. A widely studied type of SLPs are SLPs over a free monoid $(\Sigma^*, \cdot, \varepsilon, (a)_{a\in\Sigma})$, where $\cdot$ is the concatenation operator (which, as usual, is not written explicitly in expressions) and the empty string ε and every alphabet symbol $a \in \Sigma$ are added as constants. We use the term *string straight-line programs* (SSLPs for short) for these SLPs. If we want to emphasize the alphabet Σ, we speak of an SSLP over Σ. In many papers, SSLPs are just called straight-line programs; see [14] for a survey. Occasionally we consider SSLPs without a start variable S and then write (V, ρ).

Example 1. Consider the SSLP $G = (\{S, A, B, C\}, S, \rho)$ over the alphabet $\{a, b\}$ with $\rho(S) = AAB$, $\rho(A) = CBB$, $\rho(B) = CaC$, $\rho(C) = b$. We have $[\![B]\!]_G = bab$, $[\![A]\!]_G = bbabbab$, and $[\![G]\!] = bbabbabbbabbabbab$. The size of G is 8 (six concatenation operators are used in the right-hand sides, and there are two occurrences of constants).

In the next two sections, we introduce two types of algebras for trees and forests.

3 Forest algebras and forest straight-line programs

Trees and forests. Let us fix a finite set Σ of node labels for the rest of the paper. We consider Σ-labelled rooted ordered trees, where "ordered" means that the children of a node are totally ordered. Every node has a label from Σ. Note that we make no rank assumption: the number of children of a node (also called its degree) is not determined by its node label. The set of nodes (resp. edges) of t is denoted by $V(t)$ (resp., $E(t)$). A *forest* is a (possibly empty) sequence of trees. The size $|f|$ of a forest is the total number of nodes in f. The set of all Σ-labelled forests is denoted by $\mathcal{F}_0(\Sigma)$ and the set of all Σ-labelled trees is denoted by $\mathcal{T}_0(\Sigma)$. As usual, we can identify trees with expressions built up from symbols in Σ and parentheses. Formally, $\mathcal{F}_0(\Sigma)$ and $\mathcal{T}_0(\Sigma)$ can be inductively defined as the following sets of strings over the alphabet $\Sigma \cup \{(,)\}$.

- If $t_1, \ldots, t_n$ are Σ-labelled trees with $n \geq 0$, then the string $t_1 t_2 \cdots t_n$ is a Σ-labelled forest (in particular, the empty string ε is a Σ-labelled forest).
- If f is a Σ-labelled forest and $a \in \Sigma$, then $a(f)$ is a Σ-labelled tree (where the singleton tree $a()$ is usually written as a).

Let us fix a distinguished symbol $x \notin \Sigma$ for the rest of the paper (called the parameter). The set of forests $f \in \mathcal{F}_0(\Sigma \cup \{x\})$ such that x has a unique occurrence in f and this occurrence is at a leaf node is denoted by $\mathcal{F}_1(\Sigma)$. Let $\mathcal{T}_1(\Sigma) = \mathcal{F}_1(\Sigma) \cap \mathcal{T}_0(\Sigma \cup \{x\})$. Elements of $\mathcal{T}_1(\Sigma)$ (resp., $\mathcal{F}_1(\Sigma)$) are called tree contexts (resp., forest contexts). We finally define $\mathcal{F}(\Sigma) = \mathcal{F}_0(\Sigma) \cup \mathcal{F}_1(\Sigma)$ and $\mathcal{T}(\Sigma) = \mathcal{T}_0(\Sigma) \cup \mathcal{T}_1(\Sigma)$. Following [4], we define the *forest algebra* $\mathsf{FA}(\Sigma) = (\mathcal{F}(\Sigma), \boxdot, \boxplus, (a)_{a\in\Sigma}, \varepsilon, x)$ as follows:

- $\boxdot$ is the horizontal concatenation operator: for forests $f_1, f_2 \in \mathcal{F}(\Sigma)$, $f_1 \boxdot f_2$ is defined if $f_1 \in \mathcal{F}_0(\Sigma)$ or $f_2 \in \mathcal{F}_0(\Sigma)$ and in this case we set $f_1 \boxdot f_2 = f_1 f_2$ (i.e., we concatenate the corresponding sequences of trees).
- $\boxplus$ is the vertical concatenation operator: for forests $f_1, f_2 \in \mathcal{F}(\Sigma)$, $f_1 \boxplus f_2$ is defined if $f_1 \in \mathcal{F}_1(\Sigma)$ and in this case $f_1 \boxplus f_2$ is obtained by replacing in f_1 the unique occurrence of the parameter x by the forest f_2.

- Every $a \in \Sigma$ is identified with the unary function $a : \mathcal{F}(\Sigma) \to \mathcal{T}(\Sigma)$ that produces $a(f)$ when applied to $f \in \mathcal{F}(\Sigma)$.
- $\varepsilon \in \mathcal{F}_0(\Sigma)$ and $x \in \mathcal{F}_1(\Sigma)$ are constants of the forest algebra.

For better readability, we also write $f\langle g\rangle$ instead of $f \boxplus g$, fg instead of $f \boxdot g$, and a instead of $a(\varepsilon)$. Note that a forest $f \in \mathcal{F}(\Sigma)$ can be also viewed as an algebraic expression over $\mathsf{FA}(\Sigma)$, which evaluates to f itself (analogously to the free term algebra).

First-child/next-sibling encoding. The first-child/next-sibling encoding transforms a forest over some alphabet Σ into a binary tree over $\Sigma \uplus \{\bot\}$. We define $\mathrm{fcns}\colon \mathcal{F}_0(\Sigma) \to \mathcal{T}_0(\Sigma \uplus \{\bot\})$ inductively by: (*i*) $\mathrm{fcns}(\varepsilon) = \bot$ and (*ii*) $\mathrm{fcns}(a(f)g) = a(\mathrm{fcns}(f)\mathrm{fcns}(g))$ for $f, g \in \mathcal{F}_0(\Sigma)$, $a \in \Sigma$. Thus, the left (resp., right) child of a node in $\mathrm{fcns}(f)$ is the first child (resp., right sibling) of the node in f or a $\bot$-labelled leaf if it does not exist.

Example 2. If $f = a(bc)d(e)$ then

$$\begin{aligned}\mathrm{fcns}(f) &= \mathrm{fcns}(a(bc)d(e)) = a(\mathrm{fcns}(bc)\mathrm{fcns}(d(e)))\\ &= a(b(\bot\mathrm{fcns}(c))d(\mathrm{fcns}(e)\bot)) = a(b(\bot c(\bot\bot))d(e(\bot\bot)\bot)).\end{aligned}$$

Forest straight-line programs. A *forest straight-line program* over Σ, FSLP for short, is a valid straight-line program over the algebra $\mathsf{FA}(\Sigma)$ such that $[\![F]\!] \in \mathcal{F}_0(\Sigma)$. Iterated vertical and horizontal concatenations allow to generate forests, whose depth and width is exponential in the FSLP size. For an FSLP $F = (V, S, \rho)$ and $i \in \{0, 1\}$ we define $V_i = \{A \in V \mid [\![A]\!]_F \in \mathcal{F}_i(\Sigma)\}$.

Example 3. Consider the FSLP $F = (\{S, A_0, A_1, \ldots, A_n, B_0, B_1, \ldots, B_n\}, S, \rho)$ over $\{a, b, c\}$ with ρ defined by $\rho(A_0) = a$, $\rho(A_i) = A_{i-1}A_{i-1}$ for $1 \le i \le n$, $\rho(B_0) = b(A_n x A_n)$, $\rho(B_i) = B_{i-1}\langle B_{i-1}\rangle$ for $1 \le i \le n$, and $\rho(S) = B_n\langle c\rangle$. We have $[\![F]\!] = b(a^{2^n} b(a^{2^n} \cdots b(a^{2^n} c\, a^{2^n}) \cdots a^{2^n})a^{2^n})$, where b occurs 2^n many times. A more involved example can be found in the arXiv version of this paper [11].

FSLPs generalize *tree straight-line programs* (TSLPs for short) that have been used for the compression of ranked trees before, see e.g. [6, 15]. We only need TSLPs for binary trees. A TSLP over Σ can then be defined as an FSLP $T = (V, S, \rho)$ such that for every $A \in V$, $\rho(A)$ has the form a, $a(BC)$, $a(xB)$, $a(Bx)$, or $B\langle C\rangle$ with $a \in \Sigma$, $B, C \in V$. TSLPs can be used in order to compress the fcns-encoding of an unranked tree; see also [15]. It is not hard to see that an FSLP F that produces a binary tree can be transformed into a TSLP T such that $[\![F]\!] = [\![T]\!]$ and $|T| \in O(|F|)$. This is an easy corollary of our normal form for FSLPs that we introduce next (see also the proof of Proposition 9).

Normal form FSLPs. In this paragraph, we introduce a normal form for FSLPs that turns out to be crucial in the rest of the paper. An FSLP $F = (V, S, \rho)$ is in *normal form* if $V_0 = V_0^{\top} \uplus V_0^{\bot}$ and all right-hand sides have one of the following forms:

- $\rho(A) = \varepsilon$, where $A \in V_0^{\top}$,
- $\rho(A) = BC$, where $A \in V_0^{\top}, B, C \in V_0$,
- $\rho(A) = B\langle C\rangle$, where $B \in V_1$ and either $A, C \in V_0^{\bot}$ or $A, C \in V_1$,
- $\rho(A) = a(B)$, where $A \in V_0^{\bot}$, $a \in \Sigma$ and $B \in V_0$,

– $\rho(A) = a(BxC)$, where $A \in V_1$, $a \in \Sigma$ and $B, C \in V_0$.

Note that the partition $V_0 = V_0^\top \uplus V_0^\perp$ is uniquely determined by ρ. Also note that variables from V_1 produce tree contexts and variables from $V_0^\perp$ produce trees, whereas variables from $V_0^\top$ produce forests with arbitrarily many trees.

Let $F = (V, S, \rho)$ be a normal form FSLP. Every variable $A \in V_1$ produces a vertical concatenation of (possibly exponentially many) variables, whose right-hand sides have the form $a(BxC)$. This vertical concatenation is called the spine of A. Formally, we split V_1 into $V_1^\top = \{A \in V_1 \mid \exists B, C \in V_1 : \rho(A) = B\langle C\rangle\}$ and $V_1^\perp = V_1 \setminus V_1^\top$. We then define the *vertical SSLP* $F^{\square\!\!\!\!\square} = (V_1^\top, \rho_1)$ over $V_1^\perp$ with $\rho_1(A) = BC$ whenever $\rho(A) = B\langle C\rangle$. For every $A \in V_1$ the string $[\![A]\!]_{F^{\square\!\!\!\!\square}} \in (V_1^\perp)^*$ is called the *spine* of A (in F), denoted by $\mathrm{spine}_F(A)$ or just $\mathrm{spine}(A)$ if F is clear from the context. We also define the *horizontal SSLP* $F^\boxdot = (V_0^\top, \rho_0)$ over $V_0^\perp$, where ρ_0 is the restriction of ρ to $V_0^\top$. For every $A \in V_0$ we use $\mathrm{hor}(A)$ to denote the string $[\![A]\!]_{F^\boxdot} \in (V_0^\perp)^*$. Note that $\mathrm{spine}(A) = A$ (resp., $\mathrm{hor}(A) = A$) for every $A \in V_1^\perp$ (resp., $A \in V_0^\perp$).

The intuition behind the normal form can be explained as follows: Consider a tree context $t \in \mathcal{T}_1(\Sigma) \setminus \{x\}$. By decomposing t along the nodes on the unique path from the root to the x-labelled leaf, we can write t as a vertical concatenation of tree contexts $a_1(f_1 x g_1), \ldots, a_n(f_n x g_n)$ for forests $f_1, g_1, \ldots, f_n, g_n$ and symbols $a_1, \ldots, a_n$. In a normal form FSLP one would produce t by first deriving a vertical concatenation $A_1\langle\cdots\langle A_n\rangle\cdots\rangle$. Every A_i is then derived to $a_i(B_i x C_i)$, where B_i (resp., C_i) produces the forest f_i (resp., g_i). Computing an FSLP for this decomposition for a tree context that is already given by an FSLP is the main step in the proof of the normal form theorem below. Another insight is that proper forest contexts from $\mathcal{F}_1(\Sigma) \setminus \mathcal{T}_1(\Sigma)$ can be eliminated without significant size blow-up.

Theorem 4. *From a given FSLP F one can construct in linear time an FSLP F' in normal form such that $[\![F']\!] = [\![F]\!]$ and $|F'| \in O(|F|)$.*

4 Cluster algebras and top dags

In this section we introduce top dags [2, 12] as an alternative grammar-based formalism for the compression of unranked trees. A *cluster of rank* 0 is a tree $t \in \mathcal{T}_0(\Sigma)$ of size at least two. A *cluster of rank* 1 is a tree $t \in \mathcal{T}_0(\Sigma)$ of size at least two together with a distinguished leaf node that we call the *bottom boundary node* of t. In both cases, the root of t is called the *top boundary node* of t. Note that in contrast to forest contexts there is no parameter x. Instead, one of the Σ-labelled leaf nodes may be declared as the bottom boundary node. When writing a cluster of rank 1 in term representation, we underline the bottom boundary node. For instance $a(b\, c(\underline{a}\, b))$ is a cluster of rank 1. An *atomic cluster* is of the form $a(b)$ or $a(\underline{b})$ for $a, b \in \Sigma$. Let $\mathcal{C}_i(\Sigma)$ be the set of all clusters of rank $i \in \{0, 1\}$ and let $\mathcal{C}(\Sigma) = \mathcal{C}_0(\Sigma) \cup \mathcal{C}_1(\Sigma)$. We write $\mathrm{rank}(s) = i$ if $s \in \mathcal{C}_i(\Sigma)$ for $i \in \{0, 1\}$. We define the *cluster algebra* $\mathsf{CA}(\Sigma) = (\mathcal{C}(\Sigma), \odot, \oslash, (a(b), a(\underline{b}))_{a,b\in\Sigma})$ as follows:

– $\odot$ is the horizontal merge operator: $s \odot t$ is only defined if $\mathrm{rank}(s) + \mathrm{rank}(t) \leq 1$ and s, t are of the form $s = a(f)$, $t = a(g)$, i.e., the root labels coincide. Then

$s \odot t = a(fg)$. Note that at most one symbol in the forest fg is underlined. The rank of $s \odot t$ is $\mathrm{rank}(s) + \mathrm{rank}(t)$. For instance, $a(b\,c(\underline{a}\,b)) \odot a(b\,c) = a(b\,c(\underline{a}\,b)b\,c)$.
- $\oplus$ is the vertical merge operator: $s \oplus t$ is only defined if $s \in \mathcal{C}_1(\Sigma)$ and the label of the root of t (say a) is equal to the label of the bottom boundary node of s. We then obtain $s \oplus t$ by replacing the unique occurrence of $\underline{a}$ in s by t. The rank of $s \oplus t$ is $\mathrm{rank}(t)$. For instance, $a(b\,c(\underline{a}\,b)) \oplus a(b\underline{c}) = a(b\,c(a(b\underline{c})\,b))$.
- The atomic clusters $a(b)$ and $a(\underline{b})$ are constants of the cluster algebra.

A *top tree* for a tree $t \in \mathcal{T}_0$ is an algebraic expression e over the algebra $\mathsf{CA}(\Sigma)$ such that $[\![e]\!] = t$. A *top dag* over Σ is a straight-line program D over the algebra $\mathsf{CA}(\Sigma)$ such that $[\![D]\!] \in \mathcal{T}_0(\Sigma)$. In our terminology, cluster straight-line program would be a more appropriate name, but we prefer to call them top dags.

Example 5. Consider the top dag $D = (\{S, A_0, \ldots, A_n, B_0, \ldots, B_n\}, S, \rho)$, where $\rho(A_0) = b(a)$, $\rho(A_i) = A_{i-1} \odot A_{i-1}$ for $1 \le i \le n$, $\rho(B_0) = A_n \odot b(\underline{b}) \odot A_n$, $\rho(B_i) = B_{i-1} \oplus B_{i-1}$ for $1 \le i \le n$, and $\rho(S) = B_n \oplus b(c)$. We have $[\![D]\!] = b(a^{2^n} b(a^{2^n} \cdots b(a^{2^n} b(c)\, a^{2^n}) \cdots a^{2^n}) a^{2^n})$, where b occurs $2^n + 1$ many times.

5 Relative succinctness

We have now three grammar-based formalisms for the compression of unranked trees: FSLPs, top dags, and TSLPs for fcns-encodings. In this section we study their relative succinctness. It turns out that up to multiplicative factors of size $|\Sigma|$ (number of node labels) all three formalisms are equally succinct. Moreover, the transformations between the formalisms can be computed very efficiently. This allows us to transfer algorithmic results for FSLPs to top dags and TSLPs for fcns encodings, and vice versa. We start with top dags:

Proposition 6. *For a given top dag D one can compute in linear time an FSLP F such that $[\![F]\!] = [\![D]\!]$ and $|F| \in O(|D|)$.*

Proposition 7. *For a given FSLP F with $[\![F]\!] \in \mathcal{T}_0(\Sigma)$ and $|[\![F]\!]| \ge 2$ one can compute in time $O(|\Sigma| \cdot |F|)$ a top dag D such that $[\![D]\!] = [\![F]\!]$ and $|D| \in O(|\Sigma| \cdot |F|)$.*

The following example shows that the size bound in Proposition 7 is sharp:

Example 8. Let $\Sigma = \{a, a_1, ..., a_\sigma\}$ and for $n \ge 1$ let $t_n = a(a_1(a^m) \cdots a_\sigma(a^m))$ with $m = 2^n$. For every $n > \sigma$ the tree t_n can be produced by an FSLP of size $O(n)$: using $n = \log m$ many variables we can produce the forest a^m and then $O(n)$ many additional variables suffice to produce t_n. On the other hand, every top dag for t_n has size $\Omega(\sigma \cdot n)$: consider a top tree e that evaluates to t_n. Then e must contain a subexpression e_i that evaluates to the subtree $a_i(a^m)$ ($1 \le i \le \sigma$) of t_n. The subexpression e_i has to produce $a_i(a^m)$ using the $\odot$-operation from copies of $a_i(a)$. Hence, the expression for $a_i(a^m)$ has size $n = \log_2 m$ and different e_i contain no identical subexpressions. Therefore every top dag for t_n has size at least $\sigma \cdot n$.

In contrast, FSLPs and TSLPs for fcns-encodings turn out to be equally succinct up to constant factors:

Proposition 9. *Let $f \in \mathcal{F}(\Sigma)$ be a forest and let F be an FSLP (or TSLP) over $\Sigma \uplus \{\bot\}$ with $[\![F]\!] = \mathit{fcns}(f)$. Then we can transform F in linear time into an FSLP F' over Σ with $[\![F']\!] = f$ and $|F'| \in O(|F|)$.*

Proposition 10. *For every FSLP F over Σ, we can construct in linear time a TSLP T over $\Sigma \cup \{\bot\}$ with $[\![T]\!] = \mathit{fcns}([\![F]\!])$ and $|T| \in O(|F|)$.*

Proposition 10 and the construction from [7, Proposition 8.3.2] allow to reduce the evaluation of forest automata on FSLPs (for a definition of forest and tree automata, see [7]) to the evaluation of ordinary tree automata on binary trees. The latter problem can be solved in polynomial time [18], which yields:

Corollary 11. *Given a forest automaton A and an FSLP (or top dag) F we can check in polynomial time whether A accepts $[\![F]\!]$.*

In [2], a linear time algorithm is presented that constructs from a tree of size n with σ many node labels a top dag of size $O(n/\log_\sigma^{0.19} n)$. In [12] this bound was improved to $O(n \log\log n/\log_\sigma n)$ (for the same algorithm as in [2]). In [19] we recently presented an alternative construction that achieves the information-theoretic optimum of $O(n/\log_\sigma n)$ (another optimal construction was presented in [9]). Moreover, as in [2], the constructed top dag satisfies the additional size bound $O(d \cdot \log n)$, where d is the size of the minimal dag of t. With Proposition 6 and 10 we get:

Corollary 12. *Given a tree t of size n with σ many node labels, one can construct in linear time an FSLP for t (or an TSLP for $\mathit{fcns}(t)$) of size $O(n/\log_\sigma n) \cap O(d \cdot \log n)$, where d is the size of the minimal dag of t.*

6 Testing equality modulo associativity and commutativity

In this section we will give an algorithmic application which proves the utility of FSLPs (even if we deal with binary trees). We fix two subsets $\mathcal{A} \subseteq \Sigma$ (the set of *associative symbols*) and $\mathcal{C} \subseteq \Sigma$ (the set of *commutative symbols*). This means that we impose the following identities for all $a \in \mathcal{A}$, $c \in \mathcal{C}$, all trees $t_1, \ldots, t_n \in \mathcal{T}_0(\Sigma)$, all permutations $\sigma\colon \{1, \ldots, n\} \to \{1, \ldots, n\}$, and all $1 \le i \le j \le n+1$:

$$a(t_1 \cdots t_n) = a(t_1 \cdots t_{i-1} a(t_i \cdots t_{j-1}) t_j \cdots t_n) \tag{1}$$

$$c(t_1 \cdots t_n) = c(t_{\sigma(1)} \cdots t_{\sigma(n)}). \tag{2}$$

Note that the standard law of associativity for a binary symbol $\circ$ (i.e., $x \circ (y \circ z) = (x \circ y) \circ z$) can be captured by making $\circ$ an (unranked) associative symbol in the sense of (1). Our main result is:

Theorem 13. *For trees s, t we can test in polynomial time whether s and t are equal modulo the identities in* (1) *and* (2)*, if s and t are given succinctly by one of the following three formalisms: (i) FSLPs, (ii) top dags, (iii) TSLPs for the fcns-encodings of s, t.*

6.1 Associative symbols

Below, we define the associative normal form $\mathrm{nf}_{\mathcal{A}}(f)$ of a forest f and show that from an FSLP F we can compute in linear time an FSLP F' with $[\![F']\!] = \mathrm{nf}_{\mathcal{A}}([\![F]\!])$. For trees $s, t \in \mathcal{T}_0(\Sigma)$ we have that $s = t$ modulo the identities in (1) if and only if $\mathrm{nf}_{\mathcal{A}}(s) = \mathrm{nf}_{\mathcal{A}}(t)$. The generalization to forests is needed for the induction, where a slight technical problem arises. Whether the forests $t_1 \cdots t_{i-1} a(t_i \cdots t_{j-1}) t_j \cdots t_n$ and $t_1 \cdots t_n$ are equal modulo the identities in (1) actually depends on the symbol on top of these two forests. If it is an a, and $a \in \mathcal{A}$, then the two forests are equal modulo associativity, otherwise not. To cope with this problem, we use for every associative symbol $a \in \mathcal{A}$ a function $\phi_a \colon \mathcal{F}_0(\Sigma) \to \mathcal{F}_0(\Sigma)$ that pulls up occurrences of a whenever possible.

Let $\bullet \notin \Sigma$ be a new symbol. For every $a \in \Sigma \cup \{\bullet\}$ let $\phi_a \colon \mathcal{F}_0(\Sigma) \to \mathcal{F}_0(\Sigma)$ be defined as follows, where $f \in \mathcal{F}_0(\Sigma)$ and $t_1, \ldots, t_n \in \mathcal{T}_0(\Sigma)$:

$$\phi_a(b(f)) = \begin{cases} \phi_a(f) & \text{if } a \in \mathcal{A} \text{ and } a = b, \\ b(\phi_b(f)) & \text{otherwise,} \end{cases} \qquad \phi_a(t_1 \cdots t_n) = \phi_a(t_1) \cdots \phi_a(t_n).$$

In particular, $\phi_a(\varepsilon) = \varepsilon$. Moreover, define $\mathrm{nf}_{\mathcal{A}} \colon \mathcal{F}_0(\Sigma) \to \mathcal{F}_0(\Sigma)$ by $\mathrm{nf}_{\mathcal{A}}(f) = \phi_\bullet(f)$.

Example 14. Let $t = a(a(cd)b(cd)a(e))$ and $\mathcal{A} = \{a\}$. We obtain

$$\begin{aligned} \phi_a(t) &= \phi_a(a(cd)b(cd)a(e)) = \phi_a(a(cd))\phi_a(b(cd))\phi_a(a(e)) \\ &= \phi_a(cd)b(\phi_b(cd))\phi_a(e) = cdb(cd)e, \\ \phi_b(t) &= a(\phi_a(a(cd)b(cd)a(e))) = a(cdb(cd)e). \end{aligned}$$

To show the following simple lemma one considers the terminating and confluent rewriting system obtained by directing the equations (1) from right to left.

Lemma 15. *For two forests $f_1, f_2 \in \mathcal{F}_0(\Sigma)$, $\mathrm{nf}_{\mathcal{A}}(f_1) = \mathrm{nf}_{\mathcal{A}}(f_2)$ if and only if f_1 and f_2 are equal modulo the identities in* (1) *for all $a \in \mathcal{A}$.*

Lemma 16. *From a given FSLP $F = (V, S, \rho)$ over Σ one can construct in time $\mathcal{O}(|F| \cdot |\Sigma|)$ an FSLP F' with $[\![F']\!] = \mathrm{nf}_{\mathcal{A}}([\![F]\!])$.*

For the proof of Lemma 16 one introduces new variables A_a for all $a \in \Sigma \cup \{\bullet\}$ and defines the right-hand sides of F' such that $[\![A_a]\!]_{F'} = \phi_a([\![A]\!]_F)$ for all $A \in V_0$ and $[\![B_a\langle\phi_b(f)\rangle]\!]_{F'} = \phi_a([\![B\langle f\rangle]\!]_F)$ for all $B \in V_1$, $f \in \mathcal{F}_0(\Sigma)$, where b is the label of the parent node of the parameter x in $[\![B]\!]_F$. This parent node exists if we assume the FSLP F to be in normal form.

6.2 Commutative symbols

To test whether two trees over Σ are equivalent with respect to commutativity, we define a *commutative normal form* $\mathrm{nf}_{\mathcal{C}}(t)$ of a tree $t \in \mathcal{T}_0(\Sigma)$ such that $\mathrm{nf}_{\mathcal{C}}(t_1) = \mathrm{nf}_{\mathcal{C}}(t_2)$ if and only if t_1 and t_2 are equivalent with respect to the identities in (2) for all $c \in \mathcal{C}$.

We start with a general definition: Let Δ be a possibly infinite alphabet together with a total order $<$. Let $\leq$ be the reflexive closure of $<$. Define the function $\mathrm{sort}^< \colon \Delta^* \to \Delta^*$ by $\mathrm{sort}^<(a_1 \cdots a_n) = a_{i_1} \cdots a_{i_n}$ with $\{i_1, \ldots, i_n\} = \{1, \ldots, n\}$ and $a_{i_1} \leq \cdots \leq a_{i_n}$.

Lemma 17. *Let G be an SSLP over Δ and let $<$ be some total order on Δ. We can construct in time $\mathcal{O}(|\Delta| \cdot |G|)$ an SSLP G' such that $[\![G']\!] = sort^{<}([\![G]\!])$.*

In order to define the commutative normal form, we need a total order on $\mathcal{F}_0(\Sigma)$. Recall that elements of $\mathcal{F}_0(\Sigma)$ are particular strings over the alphabet $\Gamma := \Sigma \cup \{(,)\}$. Fix an arbitrary total order on Γ and let $<_{\text{llex}}$ be the *length-lexicographic order* on Γ^* induced by $<$: for $x, y \in \Gamma^*$ we have $x <_{\text{llex}} y$ if $|x| < |y|$ or ($|x| = |y|$, $x = uav$, $y = ubv'$, and $a < b$ for $u, v, v' \in \Gamma^*$ and $a, b \in \Gamma$). We now consider the restriction of $<_{\text{llex}}$ to $\mathcal{F}_0(\Sigma) \subseteq \Gamma^*$. For the proof of the following lemma one first constructs SSLPs for the strings $[\![F_1]\!], [\![F_2]\!] \in \Gamma^*$ (the construction is similar to the case of TSLPs, see [6]) and then uses [16, Lemma 3] according to which SSLP-encoded strings can be compared in polynomial time with respect to $<_{\text{llex}}$.

Lemma 18. *For two FSLPs F_1 and F_2 we can check in polynomial time whether $[\![F_1]\!] = [\![F_2]\!]$, $[\![F_1]\!] <_{\text{llex}} [\![F_2]\!]$ or $[\![F_2]\!] <_{\text{llex}} [\![F_1]\!]$.*

From the restriction of $<_{\text{llex}}$ to $\mathcal{T}_0(\Sigma) \subseteq \Gamma^*$ we obtain the function $\text{sort}^{<_{\text{llex}}}$ on $\mathcal{T}_0(\Sigma)^* = \mathcal{F}_0(\Sigma)$. We define $\text{nf}_{\mathcal{C}} \colon \mathcal{F}_0(\Sigma) \to \mathcal{F}_0(\Sigma)$ by

$$\text{nf}_{\mathcal{C}}(a(f)) = \begin{cases} a(\text{sort}^{<_{\text{llex}}}(\text{nf}_{\mathcal{C}}(f))) & \text{if } a \in \mathcal{C} \\ a(\text{nf}_{\mathcal{C}}(f)) & \text{otherwise,} \end{cases}$$

$$\text{nf}_{\mathcal{C}}(t_1 \cdots t_n) = \text{nf}_{\mathcal{C}}(t_1) \cdots \text{nf}_{\mathcal{C}}(t_n).$$

Obviously, $f_1, f_2 \in \mathcal{F}(\Sigma)$ are equal modulo the identities in (2) for all $c \in \mathcal{C}$ if and only if $\text{nf}_{\mathcal{C}}(f_1) = \text{nf}_{\mathcal{C}}(f_2)$. Using this fact and Lemma 15 it is not hard to show:

Lemma 19. *For $f_1, f_2 \in \mathcal{F}_0(\Sigma)$ we have $nf_{\mathcal{C}}(nf_{\mathcal{A}}(f_1)) = nf_{\mathcal{C}}(nf_{\mathcal{A}}(f_2))$ if and only if f_1 and f_2 are equal modulo the identities in* (1) *and* (2) *for all $a \in \mathcal{A}$, $c \in \mathcal{C}$.*

For our main technical result (Theorem 21) we need a strengthening of our FSLP normal form. Recall the notion of the *spine* from Section 3. We say that an FSLP $F = (V, S, \rho)$ is in *strong normal form* if it is in normal form and for every $A \in V_0^{\perp}$ with $\rho(A) = B\langle C \rangle$ either $B \in V_1^{\perp}$ or $|[\![C]\!]_F| \geq |[\![D]\!]_F| - 1$ for every $D \in V_1^{\perp}$ which occurs in $\text{spine}(B)$ (note that $|[\![D]\!]_F| - 1$ is the number of nodes in $[\![D]\!]_F$ except for the parameter x).

Lemma 20. *From a given FSLP $F = (V, S, \rho)$ in normal form we can construct in polynomial time an FSLP $F' = (V', S, \rho')$ in strong normal form with $[\![F]\!] = [\![F']\!]$.*

For the proof of Lemma 20 we modify the right-hand sides of variables $A \in V_0^{\perp}$ with $\rho(A) = B\langle C \rangle$ and $|\text{spine}(B)| \geq 2$. Basically, we replace the vertical concatenations $B\langle C \rangle$ by polynomially many vertical concatenations $B_i\langle C_i \rangle$ which satisfy the condition of the strong normal form. We can now prove the main technical result of this section:

Theorem 21. *From a given FSLP F we can construct in polynomial time an FSLP F' with $[\![F']\!] = nf_{\mathcal{C}}([\![F]\!])$.*

Proof. Let $F = (V, S, \rho)$. By Theorem 4 and Lemma 20 we may assume that F is in strong normal form. For every $A \in V_1$ let

$$\text{args}(A) = \{t \in \mathcal{T}_0(\Sigma) \mid |t| \geq |[\![D]\!]_F| - 1 \text{ for each symbol } D \text{ in } \text{spine}(A)\}$$

We want to construct an FSLP $F' = (V', S, \rho')$ with $V_0 \subseteq V_0'$ and $V_1 = V_1'$ such that

(1) $[\![A]\!]_{F'} = \mathrm{nf}_{\mathcal{C}}([\![A]\!]_F)$ for all $A \in V_0$,
(2) $[\![A]\!]_{F'}\langle \mathrm{nf}_{\mathcal{C}}(t)\rangle = \mathrm{nf}_{\mathcal{C}}([\![A]\!]_F\langle t\rangle)$ for all $A \in V_1, t \in \mathrm{args}(A)$.

From (1) we obtain $[\![F']\!] = [\![S]\!]_{F'} = \mathrm{nf}_{\mathcal{C}}([\![S]\!]_F) = \mathrm{nf}_{\mathcal{C}}([\![F]\!])$ which concludes the proof.

To define ρ', let $V^c = V_0^c \cup V_1^c$ with $V_1^c = \{A \in V_1 \mid \rho(A) = a(BxC) \text{ with } a \in \mathcal{C}\}$ and $V_0^c = \{A \in V_0 \mid \rho(A) = a(B) \text{ with } a \in \mathcal{C} \text{ or } \rho(A) = D\langle C\rangle \text{ with } D \in V_1^c\}$ be the set of *commutative variables*. We set $\rho'(A) = \rho(A)$ for $A \in V \setminus V^c$. For $A \in V^c$ we define $\rho'(A)$ by induction along the partial order of the dag:

1. $\rho(A) = a(B)$: Let M_A be the set of all $C \in V_0^{\perp}$ which are below A in the dag, and let $w = \mathrm{hor}(B) = [\![B]\!]_{F^{\boxdot}} \in M_A^*$. By induction, ρ' is already defined on M_A, and thus $[\![C]\!]_{F'}$ is defined for every $C \in M_A$. By Lemma 18, we can compute in polynomial time a total order $<$ on M_A such that $C < D$ implies $[\![C]\!]_{F'} \leq_{\mathrm{llex}} [\![D]\!]_{F'}$ for all $C, D \in M_A$. By Lemma 17, we can construct in linear time an SSLP $G_w = (V_w, S_w, \rho_w)$ with $[\![G_w]\!] = \mathrm{sort}^{<}(w)$, and we may assume that all variables $D \in V_w$ are new. We add these variables to V_0' together with their right hand sides $\rho'(D) = \rho_w(D)$, and we finally set $\rho'(A) = a(S_w)$.
2. $\rho(A) = B\langle C\rangle$: Let $\rho(B) = a(DxE)$. We define $G_w = (V_w, S_w, \rho_w)$ as before, but with $w = [\![DCE]\!]_{F^{\boxdot}}$ instead of $w = [\![B]\!]_{F^{\boxdot}}$, and we set $\rho'(A) = a(S_w)$.
3. $\rho(A) = a(BxC)$: We define $G_w = (V_w, S_w, \rho_w)$ as before, this time with $w = [\![BC]\!]_{F^{\boxdot}}$, and we set $\rho'(B) = a(S_w x)$.

The main idea is that the strong normal form ensures that in right-hand sides of the form $a(DxE)$ with $a \in \mathcal{C}$ one can move the parameter x to the last position (see point 3 above), since only trees that are larger than all trees produced from D and E are substituted for x. □

Proof of Theorem 13. By Proposition 6 and 9 it suffices to show Theorem 13 for the case that t_1 and t_2 are given by FSLPs F_1 and F_2, respectively. By Lemma 19 and Lemma 18 it suffices to compute in polynomial time FSLPs F_1' and F_2' for $\mathrm{nf}_{\mathcal{C}}(\mathrm{nf}_{\mathcal{A}}(t_1))$ and $\mathrm{nf}_{\mathcal{C}}(\mathrm{nf}_{\mathcal{A}}(t_2))$. This can be achieved using Lemma 16 and Theorem 21. □

7 Future work

We have shown that simple algebraic manipulations (laws of associativity and commutativity) can be carried out efficiently on grammar-compressed trees. In the future, we plan to investigate other algebraic laws. We are optimistic that our approach can be extended by idempotent symbols (meaning that $a(fttg) = a(ftg)$ for forests f, g and a tree t).

Another interesting open problem concerns context unification modulo associative and commutative symbols. The decidability of (plain) context-unification was a long standing open problem that was finally solved by Jeż [13], who showed the existence of a polynomial space algorithm. Jeż's algorithm uses his recompression technique for TSLPs. One might try to extend this technique to FSLPs with the goal of proving decidability of context unification for terms that also contain associative and commutative symbols. For first-order unification and matching [10], context matching [10], and one-context unification [8] there exist algorithms for TSLP-compressed trees that match the complexity of their uncompressed counterparts. One might also try to extend these results to the associative and commutative setting.

Acknowledgements. The first author was supported by the EPSRC grant EP/N510129/1 at the Alan Turing Institute and the EPSRC grant EP/J017728/2 at University of Edinburgh. The second author was supported by the DFG research project LO748/10-1.

References

1. S. Abiteboul, P. Bourhis, and V. Vianu. Highly expressive query languages for unordered data trees. *Theor. Comput. Syst.*, 57(4):927–966, 2015.
2. P. Bille, I. L. Gørtz, G. M. Landau, and O. Weimann. Tree compression with top trees. *Inf. Comput.*, 243:166–177, 2015.
3. A. Boiret, V. Hugot, J. Niehren, and R. Treinen. Logics for unordered trees with data constraints on siblings. In *Proc. LATA 2015*, LNCS 8977, 175–187. Springer, 2015.
4. M. Bojańczyk and I. Walukiewicz. Forest algebras. In *Proc. Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas].*, volume 2 of *Texts in Logic and Games*, 107–132. Amsterdam University Press, 2008.
5. I. Boneva, R. Ciucanu, and S. Staworko. Schemas for unordered XML on a DIME. *Theor. Comput. Syst.*, 57(2):337–376, 2015.
6. G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Information Systems*, 33(4-5):456–474, 2008.
7. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, C. Löding, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available at: http://www.grappa.univ-lille3.fr/tata, 2007.
8. C. Creus, A. Gascón, and G. Godoy. One-context unification with STG-compressed terms is in NP. In *Proc. RTA 2012*, LIPIcs 15, 149–164. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.
9. B. Dudek and P. Gawrychowski. Slowing down top trees for better worst-case bounds https://arxiv.org/abs/1801.01059, arXiv.org, 2018.
10. A. Gascón, G. Godoy, and M. Schmidt-Schauß. Unification and matching on compressed terms. *ACM Transactions on Computational Logic*, 12(4):26:1–26:37, 2011.
11. A. Gascón, M. Lohrey, S. Maneth, P. Reh, and K. Sieber. Grammar-based compression of unranked trees https://arxiv.org/abs/1802.05490, arXiv.org, 2018.
12. L. Hübschle-Schneider and R. Raman. Tree compression with top trees revisited. In *Proc. SEA 2015*, LNCS 9125, 15–27. Springer, 2015.
13. A. Jeż. Context unification is in PSPACE. In *Proc. ICALP 2014, Part II*, LNCS 8573, 244–255. Springer, 2014.
14. M. Lohrey. Algorithmics on SLP-compressed strings: a survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.
15. M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using RePair. *Information Systems*, 38(8):1150–1167, 2013.
16. M. Lohrey, S. Maneth, and F. Peternek. Compressed tree canonization. In *Proc. ICALP 2015, Part II*, 337–349. Springer, 2015.
17. M. Lohrey, S. Maneth, and C. P. Reh. Compression of unordered XML trees. In *Proc. ICDT 2017*, LIPIcs 68, 18:1–18:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.
18. 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.
19. M. Lohrey, P. Reh, and K. Sieber. Optimal top dag construction. https://arxiv.org/abs/1712.05822, arXiv.org, 2017.
20. S. Sundaram and S. K. Madria. A change detection system for unordered XML data using a relational model. *Data & Knowledge Engineering*, 72:257–284, 2012.