Parameter Reduction and Automata Evaluation for Grammar-Compressed Trees

Markus Lohrey[a], Sebastian Maneth[b], Manfred Schmidt-Schauß[c]

[a] *Universität Leipzig, Institut für Informatik, Germany*
[b] *NICTA and University of New South Wales, Australia*
[c] *Goethe-Universität Frankfurt, Institut für Informatik, Germany*

Abstract

Trees can be conveniently compressed with linear straight-line context-free tree grammars. Such grammars generalize straight-line context-free string grammars which are widely used in the development of algorithms that execute directly on compressed structures (without prior decompression). It is shown that every linear straight-line context-free tree grammar can be transformed in polynomial time into a monadic (and linear) one. A tree grammar is monadic if each nonterminal uses at most one context parameter. Based on this result, polynomial time algorithms are presented for testing whether a given (i) nondeterministic tree automaton or (ii) nondeterministic tree automaton with sibling constraints or (iii) nondeterministic tree walking automaton, accepts a tree represented by a linear straight-line context-free tree grammar. It is also shown that if tree grammars are nondeterministic or non-linear, then reducing their numbers of parameters cannot be done without an exponential blow-up in grammar size.

Keywords:

1. Introduction

The current massive increase in data volumes motivates the development of algorithms on *compressed data*, like for instance compressed strings, trees, and graphs. The general goal is to construct algorithms that work directly on compressed data, without prior decompression. Considerable amount of work has been done concerning algorithms that execute on compressed strings, see [1] for a survey. In this field, a popular succinct string representation are context-free grammars which generate exactly one string. It can

be statically guaranteed that only one string is generated, by restricting to acyclic grammars with exactly one production per nonterminal. Such grammars are known as straight-line programs, briefly SLPs. Since an SLP with n productions may generate a string of length 2^n, an SLP can be seen as a compressed representation of the generated string. Some of the nice features of SLPs are:

- Many dictionary based compression schemes, like for instance LZ78 and LZ77 can be converted efficiently into SLPs, see, e.g., [1–3].
- SLPs are based on context-free grammars and are apt for concise and clean mathematical proofs.
- For many algorithmic problems, SLPs allow efficient algorithms that avoid prior decompression. The most studied example in this context is pattern matching for compressed strings, see [4–6]. Another important example is equivalence checking of compressed strings, see [7–9].

Due to these appealing properties, it is natural to generalize SLPs to other more complex data structures. For trees, this is done in [10, 11]. There, a tree is represented by a context-free tree grammar that generates exactly one tree. Such grammars are called *straight-line context-free tree grammars*, briefly SLCF tree grammars. They generalize the sharing of repeated subtrees as well-known from DAGs (directed acyclic graphs) to the sharing of repeated patterns (a pattern is a connected subgraph of the tree) as in the sharing graphs of Lamping [12]. For tree structures of typical XML documents (i.e., the ones used for benchmarking in [10, 13, 14]), experiments show that SLCF tree grammars give approximately 2–3 times higher compression ratios [10, 13] than DAGs [14]. Note that finding a minimal SLCF tree grammar for a given tree is NP-complete (even if only linear grammars are generated; see below). The BPLEX [10] and TreeRePair [13] compressors run in linear time and approximate a minimal linear grammar (TreeRePair runs almost as fast as building the minimal unique DAG of a tree). Since sharing of patterns in an SLCF tree grammar can occur along the paths of a tree, it is possible to represent a tree of height 2^n by an SLCF grammar of size $O(n)$; this is not possible with a DAG (a DAG has the same height as its represented tree). More dramatically, an SLCF tree grammar of size $O(n)$ can even generate a full binary tree of height 2^n, which has 2^{2^n} many nodes. Hence, double exponential compression rates can be achieved.

The downside of such extreme compression capabilities is that arbitrary SLCF tree grammars do not inherit some of the nice algorithmic properties of (string) SLPs. For instance, whereas evaluating a given automaton on an SLP representation of a string can be done in polynomial time [1], this problem becomes PSPACE-complete for tree automata and SLCF tree grammars [11]. This motivates the investigation of restricted classes of SLCF tree grammars. Linearity is one of these restrictions: a context-free tree grammar is *linear* if every context parameter occurs at most once in every right-hand side. In fact, the grammars for XML document trees mentioned above are linear (both compressors BPLEX and TreeRePair generate only linear SLCF tree grammars). For linear SLCF grammars, equivalence can be checked in polynomial time [10, 15], thus generalizing the corresponding result for string SLPs by Plandowski [7] mentioned above. It is an open problem whether for non-linear SLCF tree grammars equivalence can be checked in polynomial time as well. Linear SLCF grammars have been used as structural indexes for XML querying [16, 17] and as means to speed up unification [18].

Another useful restriction on SLCF tree grammars is *k-boundedness*: a context-free tree grammar is k-bounded if every nonterminal uses at most k context parameters; 1-bounded grammars are also called *monadic*. In this paper we study the impact of the various restrictions on SLCF tree grammars with respect to compression. Our main result is the following: a given linear SLCF tree grammar can be transformed in polynomial time into an equivalent linear and monadic SLCF tree grammar (Theorem 10). In other words, for the purpose of compression by linear grammars, one parameter is already enough; the corresponding linear monadic grammars offer the same kind of compression as linear SLCF tree grammars. Linear monadic SLCF tree grammars are also used in [19–22], where they are called *singleton tree grammars*. We present three algorithmic applications of Theorem 10: it can be tested in polynomial time whether a given tree automaton accepts the tree represented by a linear SLCF tree grammar (Corollary 12). This solves our main open problem from [11], where we could only present a polynomial time algorithm for linear k-bounded SLCF tree grammars (when k is a fixed constant). Our second application generalizes Corollary 12 to tree automata with equality and disequality constraints between sibling nodes [23, 24] (Theorem 13). These are bottom-up tree automata which can test whether the subtrees rooted at children of the current node are equal or not equal. Their recognized languages are closed under Boolean operations and are strictly more general than regular tree languages (for a recent generalization see [25]).

The running time of this second polynomial time algorithm is much worse than the running time stated in Corollary 12 for ordinary tree automata; therefore we state the two results separately. Finally, we show that also nondeterministic tree walking automata can be evaluated in polynomial time over trees represented by linear SLCF tree grammar (Theorem 15). Tree walking automata process the input tree sequentially and thereby can walk up and down in the tree. Although nondeterministic tree walking automata are strictly less powerful than ordinary tree automata [26], the transformation from a nondeterministic tree walking automaton into an ordinary tree automaton requires an exponential blow-up, see, e.g., [27]. We also prove that the evaluation problem for tree walking automata with pebbles [27, 28] over trees represented by linear SLCF tree grammars (and in fact even DAGs) is PSPACE-complete (Theorem 16).

In Section 8 we show that Theorem 10 does not extend to larger classes of grammars. First, we consider *nondeterministic* linear SLCF tree grammars, i.e., acyclic grammars (no recursion) which may have several productions for each nonterminal. Such grammars represent finite sets of trees. We give an example of a linear and n-bounded nondeterministic SLCF tree grammar for which every equivalent k-bounded such grammar ($k < n$) must be exponentially larger. Using a straightforward extension of our proof of Theorem 10, we show that this exponential blow-up is also the worst case. Next, we consider *non-linear* SLCF tree grammars. We present an example of a non-linear n-bounded SLCF tree grammar of size $O(n)$ for which every equivalent k-bounded SLCF tree grammar ($k < n$) has size at least 2^{n-k}.

A preliminary version of this paper (containing the main result and its application to ordinary tree automata) appeared in [29].

2. SLCF String Grammars

For further details on context free grammars see e.g. [30]. A *straight-line context free string grammar (SLCF string grammar)* is a context free grammar $\mathcal{G} = (N, \Sigma, P, S)$ (where N is the set of nonterminals, Σ is the set of terminals, $P \subseteq N \times (N \cup \Sigma)^*$ is the set of productions, and $S \in N$ is the start nonterminal) such that the following holds:

(i) for every $A \in N$ there is *exactly one* production $(A \to w_A) \in P$ with left-hand side A and

(ii) the relation $\{(A, B) \in N \times N \mid B \text{ occurs in } w_A\}$ is acyclic.

These two conditions ensure that the language generated by $\mathcal{G}$ consists of exactly one string in Σ^*, which we denote by $\mathrm{val}(\mathcal{G})$. SLCF string grammars are also known as straight-line programs, see [1] for more details. The following simple lemma collects some algorithmic properties of SLCF string grammars. For a string $w = a_1 a_2 \cdots a_n$ and two positions $1 \leq i \leq j \leq n$, we define $|w| = n$, $w[i] = a_i$, and $w[i : j] = a_i \cdots a_j$.

Lemma 1. *Let $\mathcal{G}$ be an SLCF string grammar. There exist polynomial time algorithms for the following problems:*

- *Compute the length* $|\mathrm{val}(\mathcal{G})|$.
- *Given a position* $1 \leq i \leq |\mathrm{val}(\mathcal{G})|$, *compute the symbol* $\mathrm{val}(\mathcal{G})[i]$.
- *Given two positions* $1 \leq i \leq j \leq |\mathrm{val}(\mathcal{G})|$, *compute an SLCF string grammar* $\mathcal{H}$ *such that* $\mathrm{val}(\mathcal{H}) = \mathrm{val}(\mathcal{G})[i : j]$.

The proof of the previous lemma is folklore: the grammar is simply traversed bottom-up in one pass, while computing the lengths of the strings generated by the nonterminals. A more difficult result was shown by Plandowski [7]: It can be checked in polynomial time, whether $\mathrm{val}(\mathcal{G}) = \mathrm{val}(\mathcal{H})$ for two given SLCF string grammars $\mathcal{G}$ and $\mathcal{H}$.

W.l.o.g. we will only consider SLCF string grammars in *Chomsky normal form (CNF)*, which means that all productions are of the form $A \to a$ or $A \to BC$ for nonterminals A, B, C and a terminal a. Note that it is well-known that every context-free grammar can be transformed into CNF in polynomial time, see, e.g., [31].

In the next section we will introduce SLCF tree grammars, which generalize SLCF string grammars to trees.

3. Trees and SLCF Tree Grammars

We assume the reader to be familiar with basic tree language theory, see, e.g., [24, 32]. The following are standard definitions of labeled, ordered trees. By $\mathbb{N}$ we denote the set of natural numbers, and by $\mathbb{N}^*$ the set of finite words (sequences) over elements of $\mathbb{N}$. A *ranked alphabet* is a pair $(\mathbb{F}, \mathrm{rank})$, where $\mathbb{F}$ is a finite set of function symbols and $\mathrm{rank} : \mathbb{F} \to \mathbb{N}$ assigns to each $\alpha \in \mathbb{F}$ its rank. Let $\mathbb{F}_i = \{\alpha \in \mathbb{F} \mid \mathrm{rank}(\alpha) = i\}$ and $\mathbb{F}_{\geq i} = \bigcup_{j \geq i} \mathbb{F}_j$. Symbols in $\mathbb{F}_0$ are called *constants*. We fix a ranked alphabet $(\mathbb{F}, \mathrm{rank})$ in

the following. An $\mathbb{F}$-*labeled ordered tree* t (or *ground term* over $\mathbb{F}$) is a pair $t = (\mathrm{dom}_t, \lambda_t)$, where (i) $\mathrm{dom}_t \subseteq \mathbb{N}^*$ is finite, (ii) $\lambda_t : \mathrm{dom}_t \to \mathbb{F}$, (iii) if $w = vv' \in \mathrm{dom}_t$, then also $v \in \mathrm{dom}_t$, and (iv) if $v \in \mathrm{dom}_t$ and $\lambda_t(v) \in \mathbb{F}_n$, then $vi \in \mathrm{dom}_t$ if and only if $1 \leq i \leq n$. The edge relation of t is implicitly given as $\{(v, vi) \in \mathrm{dom}_t \times \mathrm{dom}_t \mid v \in \mathbb{N}^*, i \in \mathbb{N}\}$. Thus, $\varepsilon \in \mathrm{dom}_t$ represents the root node of t (which is labeled $\lambda_t(\varepsilon)$), and vi represents the i-th child of v. The size of t, denoted by $|t|$, is defined as $|\mathrm{dom}_t|$. We identify an $\mathbb{F}$-labeled tree t with a term in the usual way: if $\lambda_t(\varepsilon) = \alpha \in \mathbb{F}_i$, then this term is $\alpha(t_1, \ldots, t_i)$, where t_j is the term associated with the subtree of t rooted at node j. The set of all $\mathbb{F}$-labeled trees is denoted $T(\mathbb{F})$. Let us fix a countable set $\mathbb{Y} = \{y_1, y_2, \ldots\}$ of *(formal context-) parameters* (below we also use a distinguished parameter $z \notin \mathbb{Y}$). The set of all $\mathbb{F}$-labeled trees with parameters from $Y \subseteq \mathbb{Y}$ is $T(\mathbb{F}, Y)$. Formally, we consider parameters as new constants and define $T(\mathbb{F}, Y) = T(\mathbb{F} \cup Y)$. The tree $t \in T(\mathbb{F}, Y)$ is *linear*, if every parameter $y \in Y$ occurs at most once in t. For trees $t \in T(\mathbb{F}, \{y_1, \ldots, y_n\})$, $t_1, \ldots, t_n \in T(\mathbb{F}, Y)$, by $t[y_1/t_1 \cdots y_n/t_n]$ we denote the tree that is obtained by replacing in t every y_i-labeled leaf with t_i $(1 \leq i \leq n)$. A *context* is a tree $C \in T(\mathbb{F}, \mathbb{Y} \cup \{z\})$, in which the distinguished parameter z appears exactly once. Instead of $C[z/t]$ we write briefly $C[t]$. When talking about algorithms on trees, we assume the RAM model of computation, and we assume that trees are given as standard pointer representation.

For further consideration, let us fix a countable infinite set $\mathbb{N}_i$ of symbols of rank i with $\mathbb{F}_i \cap \mathbb{N}_i = \emptyset$. Hence, every finite subset $N \subseteq \bigcup_{i \geq 0} \mathbb{N}_i$ is a ranked alphabet. A *context-free tree grammar* (*over* $\mathbb{F}$) is a triple $\mathcal{G} = (N, P, S)$, where

(i) $N \subseteq \bigcup_{i \geq 0} \mathbb{N}_i$ is a finite set of *nonterminals*,

(ii) P (the set of *productions*) is a finite set of pairs of the form $(A \to t)$, where $A \in N$ and $t \in T(\mathbb{F} \cup N, \{y_1, \ldots, y_{\mathrm{rank}(A)}\})$, and

(iii) $S \in N \cap \mathbb{N}_0$ is the *start nonterminal* of rank 0.

We assume that every nonterminal $B \in N \setminus \{S\}$ as well as every terminal symbol from $\mathbb{F}$ occurs in the right-hand side t of some production $(A \to t) \in P$. For a production $(A \to t) \in P$ with $A \in N \cap \mathbb{N}_n$, we also write $A(y_1 \ldots, y_n) \to t$ in order to emphasize that $\mathrm{rank}(A) = n$. The *size* $|\mathcal{G}|$ of $\mathcal{G}$ is $|\mathcal{G}| = \sum_{(A \to t) \in P} |t|$. Let us define the derivation relation $\Rightarrow_{\mathcal{G}}$ on $T(\mathbb{F} \cup N, \mathbb{Y})$ as follows: $s \Rightarrow_{\mathcal{G}} s'$ if there exist a production $(A \to t) \in P$ with $\mathrm{rank}(A) = n$,

a context $C \in T(\mathbb{F} \cup N, \mathbb{Y} \cup \{z\})$, and trees $t_1, \ldots, t_n \in T(\mathbb{F} \cup N, \mathbb{Y})$ such that $s = C[A(t_1, \ldots, t_n)]$ and $s' = C[t[y_1/t_1 \cdots y_n/t_n]]$. The *language defined by* $\mathcal{G}$, denoted by $L(\mathcal{G})$, is the set $\{t \in T(\mathbb{F}) \mid S \Rightarrow_{\mathcal{G}}^* t\} \subseteq T(\mathbb{F})$.

As an example, consider a context-free tree grammar with the three productions $S \to A(a)$, $A(y_1) \to A(A(y_1))$, and $A(y_1) \to f(y_1, y_1)$. It should be clear that the language defined by this grammar consists of all full binary trees over the the binary symbol f and the constant symbol a.

We consider several subclasses of context-free tree grammars:

- $\mathcal{G}$ is *linear*, if for every production $(A \to t) \in P$ the term t is linear in the parameters, i.e., each element of $\{y_1, \ldots, y_{\mathrm{rank}(A)}\}$ occurs at most once in t.
- $\mathcal{G}$ is *non-deleting*, if for every production $(A \to t) \in P$, each of the parameters $y_1, \ldots, y_{\mathrm{rank}(A)}$ appears in t.
- $\mathcal{G}$ is *non-erasing*, if $t \notin \mathbb{Y}$ for every production $(A \to t) \in P$.
- $\mathcal{G}$ is *productive*, if it is non-erasing and non-deleting.
- $\mathcal{G}$ is *k-bounded* (for $k \in \mathbb{N}$), if $\mathrm{rank}(A) \leq k$ for every $A \in N$.
- $\mathcal{G}$ is *monadic* if it is 1-bounded.

Finally, a *straight-line context-free tree grammar* (*SLCF tree grammar*) is a context-free tree grammar $\mathcal{G} = (N, P, S)$, where

(i) for every $A \in N$ there is *exactly one* production $(A \to t_A) \in P$ with left-hand side A and

(ii) the relation $\{(A, B) \in N \times N \mid B \text{ occurs in } t_A\}$ is acyclic; we call the reflexive transitive closure of this relation the *hierarchical order* of $\mathcal{G}$.

Conditions (i) and (ii) ensure that $L(\mathcal{G})$ contains exactly one tree in $T(\mathbb{F})$; this tree is denoted $\mathrm{val}(\mathcal{G})$. Alternatively, for every term $t \in T(\mathbb{F} \cup N, \{y_1, \ldots, y_n\})$ we can define a term $\mathrm{val}_{\mathcal{G}}(t) \in T(\mathbb{F}, \{y_1, \ldots, y_n\})$ by induction on the hierarchical order of $\mathcal{G}$ as follows, where $1 \leq i \leq n$, $f \in \mathbb{F}_m$, and $A \in N \cap \mathbb{N}_m$:

- $\mathrm{val}_{\mathcal{G}}(y_i) = y_i$
- $\mathrm{val}_{\mathcal{G}}(f(t_1, \ldots, t_m)) = f(\mathrm{val}_{\mathcal{G}}(t_1), \ldots, \mathrm{val}_{\mathcal{G}}(t_m))$

- $\mathrm{val}_{\mathcal{G}}(A(t_1,\ldots,t_m)) = \mathrm{val}_{\mathcal{G}}(t_A)[y_1/\mathrm{val}_{\mathcal{G}}(t_1)\cdots y_m/\mathrm{val}_{\mathcal{G}}(t_m)]$.

Finally, let $\mathrm{val}_{\mathcal{G}}(A) = \mathrm{val}_{\mathcal{G}}(A(y_1,\ldots,y_{\mathrm{rank}(A)}))$ and $\mathrm{val}(\mathcal{G}) = \mathrm{val}_{\mathcal{G}}(S)$. An SLCF tree grammar can be also seen as a *recursive program scheme* [33] that generates a finite tree. SLCF tree grammars generalize SLCF *string* grammars in a natural way to trees. The following example shows that SLCF tree grammars may lead to doubly exponential compression ratios; thus, they can be exponentially more succinct than DAGs.

Example 2. *Let the (non-linear) monadic SLCF tree grammar $\mathcal{G}_n$ consist of the productions*

$$\begin{array}{lcll} S & \to & A_0(a) & \\ A_i(y_1) & \to & A_{i+1}(A_{i+1}(y_1)) & \text{for } 0 \le i < n \\ A_n(y_1) & \to & f(y_1,y_1). & \end{array}$$

Then $\mathrm{val}(\mathcal{G}_n)$ *is a complete binary tree of height* 2^n+1*. Thus,* $|\mathrm{val}(\mathcal{G}_n)| = 2\cdot 2^{2^n}-1$.

On the other hand, it is not difficult to show that for a *linear* SLCF tree grammar $\mathcal{G}$ it holds that $|\mathrm{val}(\mathcal{G})| \le 2^{O(|\mathcal{G}|)}$. Thus, linear SLCF tree grammars have at most exponential compression ratios, just like DAGs, which are the same as 0-bounded SLCF tree grammars. But even linear SLCF tree grammars can be exponentially more succinct than DAGs: the linear SLCF tree grammar $\mathcal{G}'_n$ with the productions $S \to A_0(a)$, $A_i(y_1) \to A_{i+1}(A_{i+1}(y_1))$ for $0 \le i < n$, and $A_n(y_1) \to f(y_1)$ generates a monadic tree of height 2^n+1. The minimal DAG for this tree is the tree itself and thus has size 2^n+1. The following result was shown in [10].

Proposition 3. *There exists a polynomial time algorithm that tests for two given linear SLCF tree grammars $\mathcal{G}$ and $\mathcal{H}$, whether* $\mathrm{val}(\mathcal{G}) = \mathrm{val}(\mathcal{H})$.

It is open whether Proposition 3 can be generalized to non-linear SLCF tree grammars. In [11] we could only prove a PSPACE upper bound for the equality problem for non-linear SLCF tree grammars.

The following lemma can be shown by a simple bottom-up computation of tree sizes.

Lemma 4. *For a given linear SLCF tree grammar $\mathcal{G}$, one can compute the size* $|\mathrm{val}(\mathcal{G})|$ *in polynomial time.*

4. Tree Automata

In this section we introduce various models of tree automata. We start with ordinary nondeterministic tree automata. Let us fix a ranked alphabet $\mathbb{F}$. A *nondeterministic tree automaton* over $\mathbb{F}$, NTA for short, is a tuple $\mathcal{A} = (Q, \Delta, F)$, where

(i) Q is a finite set of *states*,

(ii) $F \subseteq Q$ is the set of *final states*, and

(iii) Δ is a set of *transitions* of the form $(q_1, \ldots, q_{\mathrm{rank}(f)}, f, q)$, where $f \in \mathbb{F}$ and $q_1, \ldots, q_{\mathrm{rank}(f)}, q \in Q$.

We define the mapping $\widetilde{\Delta} : T(\mathbb{F}) \to 2^Q$ inductively as follows, where $n \geq 0$, $f \in \mathbb{F}_n$, and $t_1, \ldots, t_n \in T(\mathbb{F})$:

$$\widetilde{\Delta}(f(t_1, \ldots, t_n)) = \{q \in Q \mid \exists (q_1, \ldots, q_n, f, q) \in \Delta : q_1 \in \widetilde{\Delta}(t_1), \ldots, q_n \in \widetilde{\Delta}(t_n)\}.$$

The *language defined by* $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set

$$L(\mathcal{A}) = \{t \in T(\mathbb{F}) \mid \widetilde{\Delta}(t) \cap F \neq \emptyset\}.$$

The *size* of the NTA $\mathcal{A} = (Q, \Delta, F)$ is defined as

$$|\mathcal{A}| = \sum_{(q_1, \ldots, q_n, f, q) \in \Delta} (n \cdot \log |Q| + \log |\mathbb{F}|).$$

4.1. Tree Automata with Sibling-Constraints

A *nondeterministic tree automaton with sibling-constraints* (*over* $\mathbb{F}$), NTAC for short, is a tuple $\mathcal{A} = (Q, \Delta, F)$, where Q and F are as for NTAs and Δ is a set of *transitions* of the form $(E, D, q_1, \ldots, q_{\mathrm{rank}(f)}, f, q)$, where $E, D \subseteq \{1, \ldots, \mathrm{rank}(f)\}^2$ are disjoint relations such that D is irreflexive, $f \in \mathbb{F}$, and $q_1, \ldots, q_{\mathrm{rank}(f)}, q \in Q$. The relation E (resp. D) is a set of *equality* (resp. *disequality*) *constraints between siblings*. We define the mapping $\widetilde{\Delta} : T(\mathbb{F}) \to 2^Q$ inductively as follows, where $n \geq 0$, $f \in \mathbb{F}_n$, and $t_1, \ldots, t_n \in T(\mathbb{F})$:

$$\widetilde{\Delta}(f(t_1, \ldots, t_n)) = \{q \in Q \mid \exists (E, D, q_1, \ldots, q_n, f, q) \in \Delta : q_1 \in \widetilde{\Delta}(t_1), \ldots, q_n \in \widetilde{\Delta}(t_n), \forall (i, j) \in E : t_i = t_j, \forall (i, j) \in D : t_i \neq t_j\}.$$

The *language defined by* $\mathcal{A}$ is $L(\mathcal{A}) = \{t \in T(\mathbb{F}) \mid \widetilde{\Delta}(t) \cap F \neq \emptyset\}$. The size of the NTAC $\mathcal{A}$ is

$$|\mathcal{A}| = \sum_{(E,D,q_1,\ldots,q_n,f)\in\Delta} (n^2 + n \cdot \log|Q| + \log|\mathbb{F}|).$$

4.2. Tree Walking Automata

A tree walking automaton (TWA) [34] accept trees by walking sequentially around the input tree until an accepting state is reached. A TWA starts its walk at the root. At each step, the TWA gets the information, whether the current node is the root or the i-th child of the parent node as well as the label of the current node. Depending on this information, the automaton can move to the parent node, to a certain child node, or stay at the current node, while changing the state (or accepting the tree). Let r be the maximal arity of a symbol in the ranked alphabet $\mathbb{F}$. For a tree $t \in T(\mathbb{F})$ and a node $v \in \mathrm{dom}_t$, we define $\mathsf{type}(v) \in \{\varepsilon\} \cup \mathbb{N}$ as follows:

$$\mathsf{type}(v) = \begin{cases} \varepsilon & \text{if } v = \varepsilon \\ i & \text{if } v \in \mathbb{N}^* i, i \in \mathbb{N} \end{cases}$$

Formally, a *nondeterministic tree walking automaton* over $\mathbb{F}$ is a tuple $\mathcal{W} = (Q, \Delta, q_0, F)$, where Q and F are as for NTAs, q_0 is the *initial state*, and Δ is a set of *transitions* of the form (p, f, i, q, d), where $p, q \in Q$, $f \in \mathbb{F}$, $i \in \{\varepsilon, 1, \ldots, r\}$, and $d \in \{\uparrow, \varepsilon, 1, \ldots, \mathrm{rank}(f)\}$. Moreover, if $d = \uparrow$, then $i \neq \varepsilon$.

Let $t \in T(\mathbb{F})$ be a tree. A *configuration* of $\mathcal{W}$ on t is a pair from $Q \times \mathrm{dom}_t$. We define the one-step computation relation $\vdash_{\mathcal{W}} \subseteq (Q \times \mathrm{dom}_t) \times (Q \times \mathrm{dom}_t)$ in the usual way: $(p, u) \vdash_{\mathcal{W}} (q, v)$ if there exists a transition $(p, \lambda_t(u), \mathsf{type}(u), q, d) \in \Delta$ such that $v = ud$ if $d \in \{\varepsilon, 1, \ldots, r\}$ and $v = u'$ with $u = u'i$ and $i = \mathsf{type}(u)$ if $d = \uparrow$. Finally, t is *accepted by* $\mathcal{W}$ denoted by $t \in L(\mathcal{W})$), if there exists a sequence $(q_0, u_0) \vdash_{\mathcal{W}} (q_1, u_1) \vdash_{\mathcal{W}} \cdots (q_{n-1}, u_{n-1}) \vdash_{\mathcal{W}} (q_n, u_n)$ such that $u_0 = \varepsilon$ and $q_n \in F$. The *size* of $\mathcal{W}$ is defined as $|\mathcal{W}| = |\Delta| \cdot (\log|Q| + \log|\mathbb{F}| + \log(r))$.

TWAs are strictly less expressive than NTAs [26]; however the transformation from a TWA into an equivalent NTA is inherently exponential (for instance mentioned in [27]). Moreover, emptiness for TWAs is EXPTIME-complete [27], whereas emptiness for NTAs can be checked in polynomial time (see, e.g., [24]). An algorithm in deterministic EXPTIME for deciding emptiness of a TWA is given in [35, Theorem 5].

5. Normal Forms for Linear SLCF Tree Grammars

In this section, we only deal with *linear* SLCF tree grammars. It is easy to see that a linear SLCF tree grammar $\mathcal{G} = (N, P, S)$ can be transformed in linear time into an equivalent linear and *non-deleting* SLCF tree grammar: if for a production $A \to t_A$ (with $\mathrm{rank}(A) = n$) the parameters $y_{i_1}, \ldots, y_{i_k} \in \{y_1, \ldots, y_n\}$ do not occur in t_A, then we can reduce the rank of A to $n - k$. Moreover, if A occurs in a right-hand side t_B at position $v \in \mathrm{dom}_{t_B}$, then we remove from t_B the subtrees rooted at positions $vi_1, \ldots, vi_k$. We now produce an equivalent non-deleting grammar in one pass through $\mathcal{G}$: starting from the leaves of the hierarchical order of $\mathcal{G}$, we reduce the rank of each nonterminal A and store with it the indexes of removed parameters (so that in later occurrences of A we know which subtrees to remove). Note that the size of the new grammar is at most $|\mathcal{G}|$.

Now, let $\mathcal{G}$ be a linear and non-deleting SLCF tree grammar. Again it is easy to see that $\mathcal{G}$ can be transformed in linear time into an equivalent linear and *productive* SLCF tree grammar: we remove each production with right hand side y_1, and apply the removed productions in all remaining right-hand sides. As before, this can be done in one pass through the grammar $\mathcal{G}$, and the resulting grammar has size at most $|\mathcal{G}|$.

The previous two constructions are essentially the same as Fischer's "argument-preserving" normal form for IO macro grammars, in the proof of [36, Theorem 3.1.10]. Macro grammars are similar to context-free tree grammars except that they generate strings. Since in an SLCF tree grammar, every nonterminal has exactly one production, it is not difficult to see that the derivation order (IO or OI, see e.g. [24] for a definition) does not matter for SLCF tree grammars. It is also known that for arbitrary linear and non-deleting context-free tree grammars the derivation order again does not matter [37].

A linear SLCF tree grammar $\mathcal{G} = (N, P, S)$ is in *Chomsky normal form* (*CNF*) if it is productive, and for every production $(A \to t_A) \in P$ with $\mathrm{rank}(A) = n$, the term t_A has one of the following two forms:

(a) $f(y_1, \ldots, y_n)$ with $f \in \mathbb{F}_n$

(b) $B(y_1, \ldots, y_{i-1}, C(y_i, \ldots, y_{j-1}), y_j, \ldots, y_n)$ with $B, C \in N$, $1 \leq i \leq j \leq n + 1$.

The proof of the following theorem is a straightforward extension of the corresponding construction for context-free string grammars. In fact, for

macro grammars, a normal form similar to CNF exists (called IO standard form in [36, Definition 3.1.7]), where the nonterminal C in the second type (b) can even be assumed to be the first argument of B (for us this does not work, because in our CNF the parameters have to occur in the order $y_1, \ldots, y_{\mathrm{rank}(A)}$ in the right-hand side for A).

Theorem 5. *Let $\mathcal{G} = (N, P, S)$ be a linear and productive SLCF tree grammar over $\mathbb{F}$ and let r be the maximal rank in $N \cup \mathbb{F}$. We can construct in time $O(r \cdot |\mathcal{G}|)$ a linear SLCF tree grammar $\mathcal{G}' = (N', P', S)$ in CNF such that $N' \supseteq N$, $|N'| \leq 2 \cdot |\mathcal{G}|$, $\mathcal{G}'$ is k'-bounded, $k' \leq 2r - 1$, and $\mathrm{val}_{\mathcal{G}'}(A) = \mathrm{val}_{\mathcal{G}}(A)$ for all $A \in N$.*

Proof. Let the SLCF tree grammar $\mathcal{G} = (N, P, S)$ be linear and productive. In a first step, we ensure that for every production $(A \to t_A) \in P$, the parameters $y_1, \ldots, y_{\mathrm{rank}(A)}$ occur in this order from left to right in the tree t_A. For this, we reorder all productions bottom-up as follows. Consider a production $A(y_1, \ldots, y_n) \to t_A$ such that all productions for nonterminals in t_A are already reordered. There exists a permutation $\rho : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that the parameters $y_1, \ldots, y_n$ occur in the order $y_{\rho(1)}, \ldots, y_{\rho(n)}$ in t_A. Then we replace the production $A(y_1, \ldots, y_n) \to t_A$ by the production

$$A(y_1, \ldots, y_n) \to t_A[y_{\rho(1)}/y_1 \cdots y_{\rho(n)}/y_n]$$

and we replace every subtree of the form $A(t_1, \ldots, t_n)$ in a right-hand side by the tree $A(t_{\rho(1)}, \ldots, t_{\rho(n)})$. Note that during this process, for every node in a right-hand side the corresponding list of child-pointers is reordered only once. Therefore, we need time $O(|\mathcal{G}|)$ for this first step and the resulting grammar has the same size as before.

In a second step, we eliminate chain productions of the form $A(y_1, \ldots, y_n) \to B(y_1, \ldots, y_n)$ with $B \in N$. We can compute in time $O(|\mathcal{G}|)$ a partial mapping $f : N \to N$ such that $f(A) = B$ if and only if $A(y_1, \ldots, y_n) \Rightarrow_{\mathcal{G}}^{+} B(y_1, \ldots, y_n)$ and the right-hand side for B is not just a single nonterminal. We then remove all chain productions from $\mathcal{G}$ and replace every occurrence of a nonterminal $A \in \mathrm{dom}(f)$ in a right-hand side by $f(A)$. Again, this step does not increase the size of the grammar.

In a third step, we add for every terminal symbol $f \in \mathbb{F}_n$ for which there does not exist a production of the form $A(y_1, \ldots, y_n) \to f(y_1, \ldots, y_n)$ a new nonterminal A_f of rank n together with the production $A_f(y_1, \ldots, y_n) \to f(y_1, \ldots, y_n)$. Then, we can replace every occurrence of f in a right-hand

side of size at least two by a nonterminal. This step increases the size of the grammar by at most $\sum_{f\in\mathbb{F}}(\mathrm{rank}(f)+1)$.

In a final step, we reduce the number of nonterminals in each right-hand side to at most two. Assume that $A(y_1,\ldots,y_n)\to t_A$ is a production such that t_A consists of at least three nonterminals. The tree t_A must be of the form

$$B(y_1,\ldots,y_i,t_1,t_2,\ldots,t_k)$$

where $i\geq 0$, $k\geq 1$, and $t_1,\ldots,t_k$ are trees such that the root of t_1 is labeled by a nonterminal C. Let $m\geq 0$ be the number of parameters that appear in t_1 (thus, $y_{i+1},\ldots,y_{i+m}$ appear in t_1 in this order) and define the substitution

$$\Psi=[y_{i+m+1}/y_{i+2},y_{i+m+2}/y_{i+3},\ldots,y_n/y_{n-m+1}].$$

If the terms $t_2,\ldots,t_k$ are all parameters (i.e., $t_A=B(y_1,\ldots,y_i,t_1,y_{i+m+1},\ldots,y_n)$) or do not exist (i.e., $\mathrm{rank}(B)=i+1$), then let $\gamma=B$; otherwise let $\gamma=D$ where D is a new nonterminal of rank $n-m+1$ with the production

$$D(y_1,\ldots,y_{n-m+1})\to B(y_1,\ldots,y_i,y_{i+1},t_2\Psi,\ldots,t_k\Psi). \tag{1}$$

Clearly, the number of nonterminals in D's right-hand side is at least one less than the number of nonterminals in t_A. If t_1 contains only one nonterminal then we set $t_1'=t_1$; otherwise, we introduce the new nonterminal E of rank m with right-hand side $t_1[y_{i+1}/y_1,\ldots,y_{i+m}/y_m]$ and let $t_1'=E(y_{i+1},\ldots,y_{i+m})$. Finally, we replace the production $A(y_1,\ldots,y_n)\to t_A$ by

$$A(y_1,\ldots,y_n)\to\gamma(y_1,\ldots,y_i,t_1',y_{i+m+1},\ldots,y_n). \tag{2}$$

Note that this step increases the size of the grammar by $n+3$, due to the production (2). We now iterate this final step until the grammar is in CNF. Note that at most $2\cdot|\mathcal{G}|$ many iterations are necessary.

The correctness of the construction can be seen as follows: if the new nonterminal D is introduced, then apply the D-production in (1) to the new right-hand side for A in (2). Since, for $1\leq\nu\leq(n-i-1)$, the $(i+1+\nu)$-th subtree of D in (2) contains $y_{i+m+\nu}$ and in D's right-hand the trees $t_2,\ldots,t_k$ appear with $y_{i+m+\nu}$ replaced by $y_{i+1+\nu}$, we obtain precisely $B(y_1,\ldots,y_i,t_1',t_2,\ldots,t_k)$. If t_1 contains only one nonterminal, then $t_1'=t_1$ which concludes the correctness proof for that case. Otherwise, $t_1'=E(y_{i+1},\ldots,y_{i+m})$ and, similarly as before, application of the E-production to t_1' gives precisely t_1.

Recall that r is the maximal rank in $\mathbb{F} \cup N$. The final grammar has size at most $|\mathcal{G}| + \sum_{f \in \mathbb{F}} (\mathrm{rank}(f) + 1) + (r+2) \cdot |\mathcal{G}| \leq (r+3)|\mathcal{G}| + (r+1) \cdot |\mathbb{F}| \in O(r \cdot |\mathcal{G}|)$ (note that $|\mathbb{F}| \leq |\mathcal{G}|$, since we assume that every terminal appears in a right-hand side). The time needed to construct the final grammar is also in $O(r \cdot |\mathcal{G}|)$. The number $|N'|$ of nonterminals in the final grammar is $\leq 4 \cdot |\mathcal{G}|$ because in each iteration of the last step we add at most two new nonterminals, and the number of iterations is at most $2 \cdot |\mathcal{G}|$. In fact, it is not difficult to see that $|N'| \leq 2 \cdot |\mathcal{G}|$ because if two new nonterminals are introduced in an iteration, then the number of nonterminals in D's right-hand side is decreased by at least two with respect to t_A. □

Note that the construction of CNF in the proof of Theorem 5 also changes the depth of the grammar. The *depth* of an SLCF grammar is the maximal length of any path in the hierarchical order of the grammar. It should be clear that the depth of the new grammar $\mathcal{G}'$ in CNF is bounded by $d \cdot h$, where d is the depth of the original grammar $\mathcal{G}$, and h is the maximal height of the right-hand side tree of any production of $\mathcal{G}$. In fact, it is bounded by the maximal sum of heights of right-hand sides of nonterminals that appear on a path of the hierarchical order of $\mathcal{G}$. In [17] some experiments are reported of transforming SLCF grammars into CNF. Their grammars were obtained by running TreeRePair [13] over typical XML document trees. In those experiments, the size of a grammar never increases by more than a factor 10 when transforming into CNF; the depth on the other hand increases considerably for certain grammars (with the largest factor around 236).

Example 6. *Consider the linear and productive SLCF tree grammar $\mathcal{G}_{ex}$ with productions $S \to X(X(a,b), X(b,a))$ and $X(y_1, y_2) \to h(i(y_1), i(y_2))$. This grammar is shown on the top right of Figure 4, together with the represented tree* $\mathrm{val}(\mathcal{G}_{ex}) = h(i(h(i(a), i(b))), i(h(i(b), i(a))))$. *Note that the size of $\mathcal{G}_{ex}$ is* 12 *while the size of the tree* $\mathrm{val}(\mathcal{G}_{ex})$ *is* 13. *We now construct an equivalent grammar in CNF, following the construction in the proof of Theorem 5: Nothing needs to be done in the first two steps, because all parameters appear in all the right-hand sides of productions, and, there are no chain productions. In the third step we add new nonterminals H, I, A, B for the terminal symbols h, i, a, b, respectively, together with these productions:*

$$\begin{array}{lcl} H(y_1, y_2) & \to & h(y_1, y_2) \\ I(y_1) & \to & i(y_1) \\ A & \to & a \\ B & \to & b \end{array}$$

Moreover, we replace all occurrences of h, i, a, b in the right-hand sides of the S- and X-productions by their corresponding nonterminals. We move to the final step. Consider the X-production which contains three nonterminals in its right-hand side. Then Ψ is the identity and we introduce the new nonterminal D of rank 2 and production $D(y_1, y_2) \to H(y_1, I(y_2))$. The new X-production becomes $X(y_1, y_2) \to D(I(y_1), y_2)$. Now only the S-production is not in CNF yet. This time we introduce D' of rank q and E of rank zero and productions $D'(y_1) \to X(y_1, X(B, A))$ and $E \to X(A, B)$. We proceed similarly and finally obtain the following grammar in CNF (plus the above displayed productions for H, I, A, B).

$$\begin{array}{lcllcl} S & \to & D'(E) & E'' & \to & D'''(B) \\ E & \to & D''(A) & D'''(y_1) & \to & X(y_1, A) \\ D''(y_1) & \to & X(y_1, B) & X(y_1, y_2) & \to & D(I(y_1), y_2) \\ D'(y_1) & \to & X(y_1, E'') & D(y_1, y_2) & \to & H(y_1, I(y_2)). \end{array}$$

As another example, consider regular tree grammars, i.e., context-free tree grammars in which all nonterminals are of rank zero: they do *not* allow for a normal form in which at most two nonterminals appear in every right-hand size. To see this, consider the grammar with the two productions $S \to f(S, S, S)$ and $S \to a$. Clearly for this language there is no regular grammar with less than three nonterminals in the right-hand side of each production. On the other hand, if we do allow parameters, then the following grammar in CNF can be given (obtained by the construction in the proof of Theorem 5):

$$\begin{array}{lcl} S & \to & B(S) \\ B(y_1) & \to & C(y_1, S) \\ C(y_1, y_2) & \to & F(y_1, y_2, S) \\ F(y_1, y_2, y_3) & \to & f(y_1, y_2, y_3) \\ S & \to & a. \end{array}$$

Linear SLCF tree grammars in CNF can be stored more efficiently than ordinary SLCF tree grammars: if we know the rank of each (non)terminal, then for a right-hand side $B(y_1, \ldots, y_i, C(y_{i+1}, \ldots, y_j), y_{j+1}, \ldots, y_m)$ (resp. $f(y_1, \ldots, y_n)$) we only need to store the triple (B, C, i) (resp. the symbol f) which has size $O(\log |N| + \log k)$ if the grammar is k-bounded and N is its set of nonterminals. We call this new representation of a CNF grammar its *triple notation*. From a given linear SLCF tree grammar $\mathcal{G}$, we can construct an equivalent linear SLCF tree grammar in CNF in time $O(r \cdot |\mathcal{G}|)$ (where r

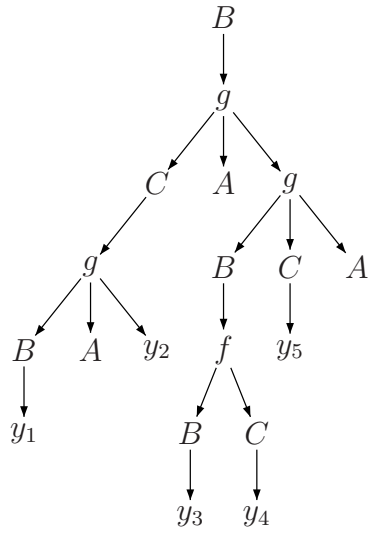


Figure 1: A skeleton tree

is again the maximal rank of (non)terminals) which only needs space $O(|\mathcal{G}| \cdot (\log |\mathcal{G}| + \log(r)))$ in triple notation.

6. Parameter Reduction in Linear SLCF Tree Grammars

In this section our main result is proved. We show that a given linear SLCF tree grammar can be made monadic in polynomial time.

A *skeleton tree* of rank $n \geq 0$ is a tree $s \in T(\mathbb{N}_0 \cup \mathbb{N}_1 \cup \mathbb{F}_{\geq 2}, \{y_1, \ldots, y_n\})$, such that every parameter y_i $(1 \leq i \leq n)$ occurs exactly once in s and the following additional properties are satisfied.

(a) The tree s does not contain a subtree of the form $X(Y(t))$ for $X, Y \in \mathbb{N}_1$.

(b) For every subtree $f(t_1, \ldots, t_m)$ of s with $f \in \mathbb{F}_{\geq 2}$ there exist at least two distinct $i \in \{1, \ldots, m\}$ such that t_i contains a parameter from $\{y_1, \ldots, y_n\}$.

Example 7. *Figure 1 shows a skeleton tree of rank 5, where* $f \in \mathbb{F}_2$, $g \in \mathbb{F}_3$, $A \in \mathbb{N}_0$ *and* $B, C \in \mathbb{N}_1$.

In our construction, a skeleton tree will store the branching structure (with respect to those leaf nodes that are parameters) of the tree generated by a

certain nonterminal, i.e., the information on how the paths from the root to parameters branch. Nonterminals of rank one in a skeleton tree represent those tree parts that are in between two branching nodes in this branching structure. The crucial point about skeleton trees is that their size can be bounded polynomially. For the following lemma, it is important that a skeleton tree only contains function symbols of rank ≥ 2.

Lemma 8. *Let r be the maximal rank of a symbol from $\mathbb{F}$. A skeleton tree s of rank $n \geq 1$ contains at most $2(r \cdot n - r + 1)$ many nodes.*

Proof. The number of nodes in s labeled with a symbol from $\mathbb{F}_{\geq 2}$ can be at most $n-1$ due to (b). From (a) it follows that the number of $\mathbb{N}_1$-labeled nodes is at most $r \cdot (n-1) + 1$. Finally, the number of leaves of s can be at most $(r-1) \cdot (n-1) + 1$. Hence, s has at most $2(r \cdot n - r + 1)$ many nodes. □

Let $\mathcal{G} = (N, P, S)$ be a linear SLCF tree grammar. By Theorem 5 we may assume that $\mathcal{G}$ is in CNF. The set of nonterminals N is a finite subset of $\bigcup_{i \geq 0} \mathbb{N}_i$. We now define in a bottom-up process, for every nonterminal A of rank $n \geq 1$, a skeleton tree sk_A of rank n. Simultaneously, we construct a new linear and monadic SLCF tree grammar $\mathcal{G}' = (N', P', S)$. Consider a production $A \to t_A$ from P and let $n = \mathrm{rank}(A)$.

Case 1. $t_A = f(y_1, \ldots, y_n)$, where $f \in \mathbb{F}_n$: if $n \leq 1$, then we add the production $A(y_1, \ldots, y_n) \to t_A$ to P' and set $\mathrm{sk}_A = A(y_1, \ldots, y_n)$. If $n \geq 2$, then we set $\mathrm{sk}_A = t_A$ and do not add any new productions to P'.

Case 2. $t_A = B(y_1, \ldots, y_{i-1}, C(y_i, \ldots, y_{j-1}), y_j, \ldots, y_n)$, where $i \leq j$ and the trees sk_B, sk_C are already constructed. In a first step define the tree

$$s = \mathrm{sk}_B[y_i/\mathrm{sk}_C[y_1/y_i, y_2/y_{i+1}, \ldots, y_{j-i}/y_{j-1}], \\ y_{i+1}/y_j, y_{i+2}/y_{j+1}, \ldots, y_{n+i-j+1}/y_n]. \quad (3)$$

But this tree is not necessarily a skeleton tree; it may violate conditions (a) and (b) on skeleton trees. Hence, we apply a contract-operation to s which yields the skeleton tree sk_A. Moreover, as a side effect, the contract-operation adds new productions and nonterminals to $\mathcal{G}'$. The contract-operation works in two steps:

Contract-1 (see Figure 2). Assume that s contains a subtree of the form $Y(Z(t))$. There can be only one subtree of this form in s. We now do the following:

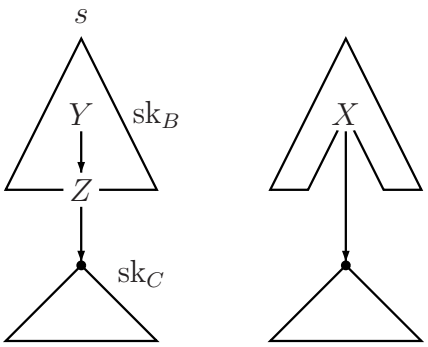


Figure 2: Contract-1

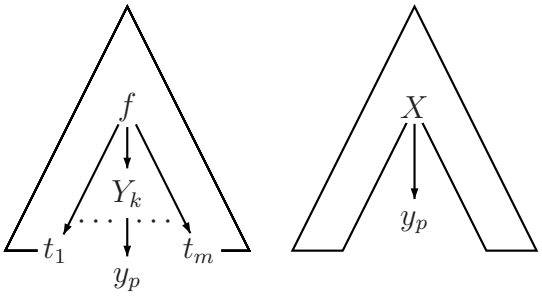


Figure 3: Contract-2

1. Add a fresh nonterminal $X \in \mathbb{N}_1$ of rank 1 to N'.
2. Add the production $X(y_1) \to Y(Z(y_1))$ to P'.
3. Replace the subtree $Y(Z(t))$ by $X(t)$.

Contract-2 (see Figure 3). After contract-1, s can only violate condition (b) for skeleton trees. Hence, assume that s contains a subtree of the form $f(t_1, \ldots, t_m)$ such that $f \in \mathbb{F}_{\geq 2}$ and there is exactly one $k \in \{1, \ldots, m\}$ such that t_k contains a parameter from $\{y_1, \ldots, y_n\}$, say y_p. Again there can be only one subtree of this form in s. Moreover, this case may only occur, if C has rank 0.

Since condition (a) is already satisfied, every subtree t_ℓ ($\ell \neq k$) is of the form Z_ℓ or $Y_\ell(Z_\ell)$ with $Y_\ell \in \mathbb{N}_1$ and $Z_\ell \in \mathbb{N}_0$, whereas t_k is either y_p or of the form $Y_k(y_p)$ for $Y_k \in \mathbb{N}_1$. We do the following:

1. Add a fresh nonterminal $X \in \mathbb{N}_1$ of rank 1 to N'.
2. Add to P' the production $X(y_1) \to f(t_1, \ldots, t_{k-1}, t_k[y_p/y_1], t_{k+1}, \ldots, t_m)$.
3. Replace the subtree $f(t_1, \ldots, t_m)$ of s by $X(y_p)$.

After this operation, another contract-1 operation might be necessary (if the new subtree $X(y_p)$ is below an $\mathbb{N}_1$-labeled node). The resulting tree is the skeleton tree sk_A. Now no more contract operations are possible.

Note that the SLCF tree grammar $\mathcal{G}'$ is linear, productive, and monadic. The following lemma can be shown by induction on the hierarchical order of $\mathcal{G}$.

Lemma 9. *For every nonterminal A of $\mathcal{G}$,* $\mathrm{val}_{\mathcal{G}}(A) = \mathrm{val}_{\mathcal{G}'}(\mathrm{sk}_A)$.

Proof. The lemma can be easily shown by induction on the hierarchical order of $\mathcal{G}$. Consider a production $(A \to t_A) \in P$ with $n = \mathrm{rank}(A)$. If the right-hand side t_A is of the form $f(y_1, \ldots, y_n)$, then we have either $\mathrm{sk}_A = A(y_1, \ldots, y_n)$ and $(A(y_1, \ldots, y_n) \to t_A) \in P'$ (if $n \leq 1$) or $\mathrm{sk}_A = f(y_1, \ldots, y_n)$. Hence, $\mathrm{val}_{\mathcal{G}}(A) = f(y_1, \ldots, y_n) = \mathrm{val}_{\mathcal{G}'}(\mathrm{sk}_A)$.

Now assume that the right-hand side t_A is of the form

$$B(y_1, \ldots, y_{i-1}, C(y_i, \ldots, y_{j-1}), y_j, \ldots, y_n),$$

where $i \leq j$ and let s be the term from (3). The productions that were added to P' during the contract-operations ensure that $\mathrm{val}_{\mathcal{G}'}(s) = \mathrm{val}_{\mathcal{G}'}(\mathrm{sk}_A)$. Hence, by induction we obtain:

$$\begin{aligned}
\mathrm{val}_{\mathcal{G}'}(\mathrm{sk}_A) &= \mathrm{val}_{\mathcal{G}'}(s) \\
&= \mathrm{val}_{\mathcal{G}'}(\mathrm{sk}_B[y_i/\mathrm{sk}_C[y_1/y_i \cdots y_{j-i}/y_{j-1}],\ y_{i+1}/y_j \cdots y_{n+i-j}/y_n]) \\
&= \mathrm{val}_{\mathcal{G}'}(\mathrm{sk}_B)[y_i/\mathrm{val}_{\mathcal{G}'}(\mathrm{sk}_C)[y_1/y_i \cdots y_{j-i}/y_{j-1}],\ y_{i+1}/y_j \cdots y_{n+i-j}/y_n] \\
&= \mathrm{val}_{\mathcal{G}'}(B)[y_i/\mathrm{val}_{\mathcal{G}'}(C)[y_1/y_i \cdots y_{j-i}/y_{j-1}],\ y_{i+1}/y_j \cdots y_{n+i-j}/y_n] \\
&= \mathrm{val}_{\mathcal{G}}(B(y_1, \ldots, y_{i-1}, C(y_i, \ldots, y_{j-1}), y_j, \ldots, y_n)) \\
&= \mathrm{val}_{\mathcal{G}}(A).
\end{aligned}$$

□

Theorem 10. *Let r be the maximal rank of a symbol from $\mathbb{F}$. From a given linear and k-bounded SLCF tree grammar $\mathcal{G} = (N, P, S)$ we can construct in time $O(k \cdot r \cdot |\mathcal{G}|)$ a linear, productive, and monadic SLCF tree grammar $\mathcal{G}' = (N', P', S)$ of size $O(r \cdot |\mathcal{G}|)$ such that $N \cap (\mathbb{N}_0 \cup \mathbb{N}_1) \subseteq N'$ and $\mathrm{val}_{\mathcal{G}'}(A) = \mathrm{val}_{\mathcal{G}}(A)$ for every $A \in N \cap (\mathbb{N}_0 \cup \mathbb{N}_1)$.*

Proof. Using the constructions from Section 5, we first transform $\mathcal{G}$ into a linear CNF grammar $\mathcal{H}$ with $O(|\mathcal{G}|)$ many nonterminals. This needs time $O(\max\{k, r\} \cdot |\mathcal{G}|)$. Now we construct for every nonterminal A of $\mathcal{H}$ the skeleton tree sk_A and simultaneously the linear and monadic SLCF tree grammar $\mathcal{H}'$. In order to construct the tree s in Equation (3), we have to copy the already constructed skeleton trees sk_B and sk_C (since we may need these trees in later steps), which by Lemma 8 needs time $O(k \cdot r)$. The construction of sk_A from s needs at most three contraction steps, each of which requires $O(1)$

many pointer operations. Moreover, in every contraction step we add to $\mathcal{H}'$ a production of size at most $O(r)$. Hence, the total size of $\mathcal{H}'$ is $O(r \cdot |\mathcal{G}|)$ and the construction takes time $O(k \cdot r \cdot |\mathcal{G}|)$. We obtain the final grammar $\mathcal{G}'$ by adding to $\mathcal{H}'$ every nonterminal $A \in N \cap (\mathbb{N}_0 \cup \mathbb{N}_1)$, which does not already belong to $\mathcal{H}'$, together with the production $A \to \mathrm{sk}_A$. By Lemma 9 we have $\mathrm{val}_{\mathcal{G}'}(A) = \mathrm{val}_{\mathcal{G}}(A)$. Note that in general $\mathcal{G}'$ is not in CNF, and that it might contain useless productions. □

Finite unions of linear monadic SLCF tree grammars are studied e.g. in [38, 39] under the name *singleton tree grammar*. They are, by Theorem 10, polynomially equivalent to finite unions of linear SLCF grammars; hence, their results can also be applied to linear grammars.

Example 11. *We transform the linear CNF grammar constructed in Example 6 into an equivalent linear monadic SLCF tree grammar. We start with the set of productions* $P' = \{A \to a, B \to b, I(y_1) \to i(y_1)\}$ *(see Case 1) and the following skeleton trees:*

$$\mathrm{sk}_A = A, \qquad \mathrm{sk}_B = B, \qquad sk_I = I(y_1), \qquad sk_H = h(y_1, y_2).$$

Next, for X *and* D *we obtain without contract operations:*

$$\mathrm{sk}_D = h(y_1, I(y_2)), \qquad \mathrm{sk}_X = h(I(y_1), I(y_2)).$$

Let us now construct $\mathrm{sk}_{D'''}$, $\mathrm{sk}_{D''}$, $\mathrm{sk}_{E''}$, sk_E, $\mathrm{sk}_{D'}$, *and* sk_S *in this order:*

- *construction of* $\mathrm{sk}_{D'''}$*: For the tree* s *in (3) we obtain* $s = h(I(y_1), I(A))$. *With contract-2, we obtain the new production* $J(y_1) \to h(I(y_1), I(A))$ *and the skeleton tree* $\mathrm{sk}_{D'''} = J(y_1)$.
- *Construction of* $\mathrm{sk}_{D''}$*: we get* $s = h(I(y_1), I(B))$. *With contract-2, we obtain the new production* $K(y_1) \to h(I(y_1), I(B))$ *and the skeleton tree* $\mathrm{sk}_{D''} = K(y_1)$.
- *Construction of* $\mathrm{sk}_{E''}$*: we get* $s = J(B)$. *Thus, we do not add a new production to* P' *and set* $\mathrm{sk}_{E''} = J(B)$.
- *Construction of* sk_E*: we get* $s = K(A)$. *Again, we do not add a new production to* P' *and set* $\mathrm{sk}_E = K(A)$.

- *Construction of* $\mathrm{sk}_{D'}$*: we get* $s = h(I(y_1), I(J(B)))$*. A first contract-1 operation adds the production* $L(y_1) \to I(J(y_1))$ *to* P' *and updates* s *to* $s = h(I(y_1), L(B))$*. Now, we have to apply another contract-2 operation, which adds the production* $M(y_1) \to h(I(y_1), L(B))$ *to* P'*. We set* $\mathrm{sk}_{D'} = M(y_1)$*.*

- *Construction of* sk_S*. We set* $s = M(K(A))$*. Hence, we add to* P' *the production* $N(y_1) \to M(K(y_1))$ *and set* $\mathrm{sk}_S = N(A)$*.*

Thus, an equivalent linear and monadic SLCF tree grammar contains the following productions:

$$\begin{array}{llllllllll} S & \to & N(A) & J(y_1) & \to & h(I(y_1), I(A)) & M(y_1) & \to & h(I(y_1), L(B)) \\ A & \to & a & K(y_1) & \to & h(I(y_1), I(B)) & N(y_1) & \to & M(K(y_1)) \\ B & \to & b & L(y_1) & \to & I(J(y_1)) & I(y_1) & \to & i(y_1). \end{array}$$

Note that the resulting monadic grammar is not in CNF. In order to make more visible the grammar change when moving from the original (binary) grammar to a monadic one, we combine some of the productions in the above grammar: we expand S*'s right-hand side by application of the* N*-production, and rename* M *into* X_0 *and* K *into* X_1*. The resulting* S *production is shown in the bottom right of Figure 4. Through similar expansions we obtain the grammar in the bottom of Figure 4. It shows that the three occurrences of* X *of the original grammar have become three different versions in the monadic grammar:* X_0*,* X_1*, and* X_2*. Note that* X_0 *generates* X_2 *in a 'recursive' way.*

7. Applications to Tree Automata Evaluation

As an immediate corollary of Theorem 10 and [11, Theorem 1] we obtain the following result:

Corollary 12. *For a given NTA* $\mathcal{A}$ *with* n *states and a given linear and* k*-bounded SLCF tree grammar* $\mathcal{G}$ *such that* r *is the maximal rank of a terminal symbol from* $\mathbb{F}$*, we can check in time* $O(r \cdot |\mathcal{G}| \cdot (k + |\mathcal{A}| \cdot n^2))$*, whether* $\mathrm{val}(\mathcal{G}) \in L(\mathcal{A})$*.*

Proof. By Theorem 10 we can transform the linear and k-bounded SLCF tree grammar $\mathcal{G}$ in time $O(r \cdot |\mathcal{G}| \cdot k)$ into a monadic linear SLCF tree grammar $\mathcal{G}$. The result follows, since by [11, Theorem 1] (where NTAs are simply

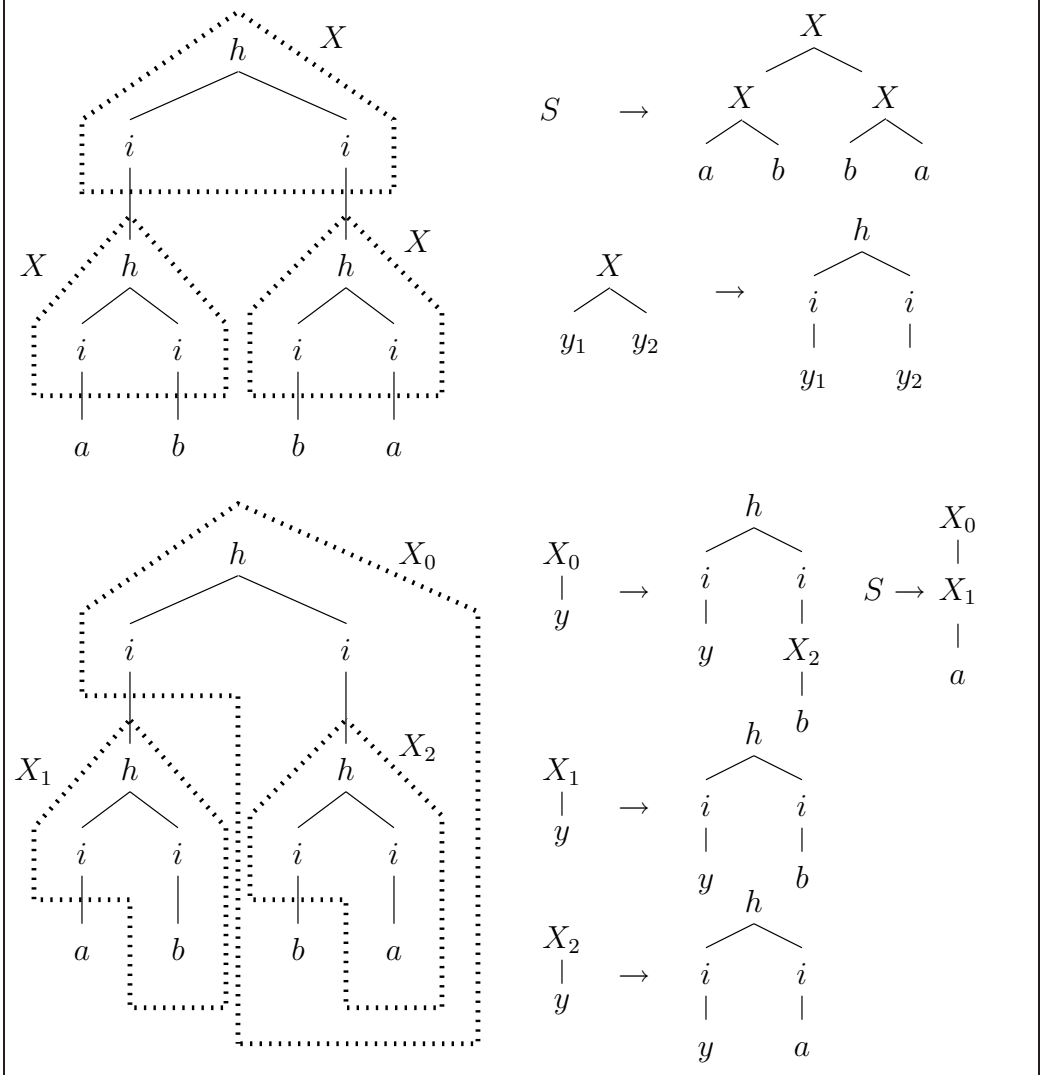


Figure 4: The grammar $\mathcal{G}_{\text{ex}}$ (top) and an equivalent monadic grammar (bottom)

called TAs) one can check for (i) a given NTA $\mathcal{A}$ with n states and (ii) a given linear and k-bounded SLCF tree grammar $\mathcal{G}$ in time $O(|\mathcal{G}| \cdot |\mathcal{A}| \cdot n^{k+1})$, whether $\mathrm{val}(\mathcal{G}) \in L(\mathcal{A})$ (if $\mathcal{A}$ is a deterministic bottom-up tree automaton then time $O(|\mathcal{G}| \cdot |\mathcal{A}| \cdot n^k)$ suffices). In order to make the paper self-contained, let us briefly explain the argument. For every nonterminal A of rank $r \leq k$ and every tuple $(q_1, \ldots, q_r, q) \in Q^{r+1}$ (Q is the set of states of $\mathcal{A}$), we compute a Boolean value $\mathsf{ok}_A(q_1, \ldots, q_r, q)$ with the following meaning: $\mathsf{ok}_A(q_1, \ldots, q_r, q) = \mathsf{true}$ if and only if there is a run of $\mathcal{A}$ on the tree $\mathrm{val}_{\mathcal{G}}(A)(y_1, \ldots, y_r)$ such that $\mathcal{A}$ enters the tree $\mathrm{val}_{\mathcal{G}}(A)(y_1, \ldots, y_r)$ at the unique occurrence of the parameter y_i in state q_i ($1 \leq i \leq r$) and arrives in state q at the root. For the nonterminal A, we have to compute $|Q|^{r+1} \leq n^{k+1}$ values. Each of these values can be computed in time $|\mathcal{A}| \cdot |t_A|$ (where t_A is the right-hand side for A), assuming that all ok-values for hierarchically smaller nonterminals are already computed. By taking the sum over all nonterminals, we obtain the time bound $O(|\mathcal{G}| \cdot |\mathcal{A}| \cdot n^{k+1})$. □

We may assume that $r, k \leq |\mathcal{G}|$ in Corollary 12, since we assume for context-free tree grammars that every (non)terminal occurs in a right-hand side. Moreover, we can eliminate states from an NTA that do not occur in transition tuples. Hence, $n \leq |\mathcal{A}|$. Thus, the time bound in Corollary 12 can be replaced by $O(|\mathcal{G}|^3 + |\mathcal{G}|^2 \cdot |\mathcal{A}|^3)$. Hence, $\mathrm{val}(\mathcal{G}) \in L(\mathcal{A})$ can be checked in polynomial time.

7.1. Tree Automata with Sibling-Constraints

In this section, we extend Corollary 12 to tree automata with sibling-constraints.

Theorem 13. *The problem of checking* $\mathrm{val}(\mathcal{G}) \in L(\mathcal{A})$ *for a given linear SLCF tree grammar* $\mathcal{G}$ *and a given NTAC* $\mathcal{A}$ *can be solved in polynomial time.*

Proof. By Theorem 10 we can assume that $\mathcal{G} = (N, P, S)$ is linear and monadic. Moreover, by introducing additional nonterminals it is easy to normalize $\mathcal{G}$ in linear time such that all productions in P are of one of the following 4 types:

- $A \to f(A_1, \ldots, A_n)$ for $A, A_1 \ldots, A_n \in \mathbb{N}_0$ and $f \in \mathbb{F}_n$
- $A \to B(C)$ for $A, C \in \mathbb{N}_0$ and $B \in \mathbb{N}_1$

- $A(y) \to f(A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n)$ for $A \in \mathbb{N}_1$, $A_1, \ldots, A_n \in \mathbb{N}_0$, $f \in \mathbb{F}_{n+1}$
- $A(y) \to B(C(y))$ for $A, B, C \in \mathbb{N}_1$

Let $\mathcal{A} = (Q, \Delta, F)$ be an NTAC. Along the hierarchical order of $\mathcal{G}$ we will compute for every $A \in \mathbb{N}_0 \cap N$ the set of states $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(A))$. At the end, we have to check whether $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(S)) \cap F \neq \emptyset$. Consider a nonterminal $A \in \mathbb{N}_0 \cap N$.

Case 1. The production for A is of the form $A \to f(A_1, \ldots, A_n)$. Assume that for every $1 \leq i \leq n$, the set of states $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(A_i))$ is already computed. Using Proposition 3, we can find out in polynomial time which of the trees $\mathrm{val}_{\mathcal{G}}(A_i)$ $(1 \leq i \leq n)$ are equal or disequal. Using this information, it is straightforward to compute the set $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(A))$.

Case 2. The production for A is of the form $A \to B(C)$. This case requires more work. Assume that the set of states $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(C))$ is already computed. Define an SLCF *string* grammar $\mathcal{G}_B$ in CNF as follows:

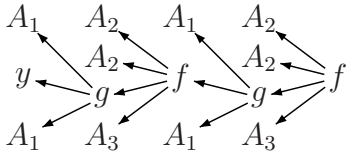
- The set of nonterminals is $\mathbb{N}_1 \cap N$, i.e., the nonterminals of $\mathcal{G}$ of rank one.
- The set of terminal symbols is $\Sigma = \{[A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n, f] \mid \exists X \in \mathbb{N}_1 \cap N : (X(y) \to f(A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n)) \in P\}$.
- If $(X(y) \to Y(Z(y))) \in P$, then $\mathcal{G}_B$ contains the production $X \to ZY$ (note that we reverse the order of the nonterminals Y and Z); if $(X(y) \to f(A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n)) \in P$, then $\mathcal{G}_B$ contains the production $X \to [A_1 \ldots, A_{i-1}, y, A_i, \ldots, A_n, f]$. These are all productions of $\mathcal{G}_B$.
- The start nonterminal of $\mathcal{G}_B$ is B.

The string generated by $\mathcal{G}_B$ represents the outcome of a partial derivation from the nonterminal B in the tree grammar $\mathcal{G}$, where the derivation process is stopped as soon as a nonterminal of rank zero is reached. Let us denote this tree with $t_B^0(y)$. This tree has a unique occurrence of the parameter y and let $p_y \in \mathbb{N}^*$ be the unique y-labeled node of $t_B^0(y)$.

Example 14. *Let $\mathcal{G}$ contain the following four productions for nonterminals of rank one: $B(y) \to B_1(B_1(y))$, $B_2(y) \to f(A_2, A_2, y, A_3)$, $B_1(y) \to B_2(B_3(y))$, $B_3(y) \to g(A_1, y, A_1)$. Here A_1, A_2, A_3 are nonterminals of rank 0. Then, the SLCF string grammar $\mathcal{G}_B$ consists of the productions $B \to B_1B_1$, $B_2 \to [A_2, A_2, y, A_3, f]$, $B_1 \to B_3B_2$, and $B_3 \to [A_1, y, A_1, g]$ and generates the string*

$$\mathrm{val}(\mathcal{G}_B) = [A_1, y, A_1, g]\,[A_2, A_2, y, A_3, f]\,[A_1, y, A_1, g]\,[A_2, A_2, y, A_3, f].$$

This string represents the following tree, which is $t^0_B(y)$:



We have $p_y = 3232$.

Let $N_{B,0} \subseteq \mathbb{N}_0 \cap N$ be the set of all nonterminals of rank 0 that appear in terminal symbols of $\mathrm{val}(\mathcal{G}_B)$. In our example, we have $N_{B,0} = \{A_1, A_2, A_3\}$.

Claim 1. The set $N_{B,0}$ can be computed in polynomial time.

We compute for every nonterminal X of $\mathcal{G}_B$ a set $N_{X,0} \subseteq \mathbb{N}_0 \cap N$. If $X \to [A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n, f]$ is a production of $\mathcal{G}_B$, then set $N_{X,0} = \{A_1, \ldots, A_n\}$. If $X \to YZ$ is a production of $\mathcal{G}_B$, then set $N_{X,0} = N_{Y,0} \cup N_{Z,0}$. In this way we can compute the set $N_{B,0}$ in polynomial time.

Recall that our goal is to compute in polynomial time the set of states $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(A))$. For this, we will work with the SLCF string grammar $\mathcal{G}_B$, which generates a string representation of the tree t^0_B. The problem is of course that the size of this tree is exponential in the worst case. Assume for a moment that $\mathcal{A}$ is an ordinary NTA (without sibling constraints). Then we could proceed as follows: First of all, if we have a terminal $a = [A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n, f]$ of $\mathcal{G}_B$ then one can associate with a a relation $R_a \subseteq Q \times Q$ with the following meaning: $(q, q') \in R_a$ if and only if $\mathcal{A}$ has a run on the tree $\mathrm{val}_{\mathcal{G}}(f(A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n))$ that enters the tree at the parameter y in state q and arrives at the root in state q'. Using the fact that the state sets $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(A_j))$ are already computed, one can compute the relation R_a easily. Then using the productions of $\mathcal{G}_B$ one can associate a relation R_X with every nonterminal of $\mathcal{G}_B$ as follows: If $X \to a$

is a production of $\mathcal{G}_B$, then set $R_X = R_a$. If $X \to ZY$ is a production of $\mathcal{G}_B$, then R_X is the composition $R_Z \circ R_Y$. The relation R_B is in some sense the semantics of the tree $t_B^0(y)$ under the NTA $\mathcal{A}$: $(q, q') \in R_B$ if and only if $\mathcal{A}$ can enter $t_B^0(y)$ at the parameter y in state q and arrive in state q' at the root. Finally, if R_B is computed, then $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(A))$ can be computed as $\{q' \in Q \mid \exists q \in \widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(C)) : (q, q') \in R_B\}$ (recall that $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(C))$ is already computed).

Unfortunately, this procedure fails in our situation, since $\mathcal{A}$ is an NTAC. Therefore, one cannot associate a relation $R_a \subseteq Q \times Q$ with a nonterminal $a = [A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n, f]$ of $\mathcal{G}_B$ as above: One has to know which tree is substituted for y in order to know which of the sibling-constraints are satisfied. But in our situation, we can solve this problem as follows: We know that the parameter y in $t_B^0(y)$ is replaced by the tree $\mathrm{val}_{\mathcal{G}}(C)$. By Proposition 3, we can check in polynomial time which of the trees $\mathrm{val}_{\mathcal{G}}(X)$ for $X \in \{C\} \cup N_{B,0}$ are equal. Moreover, the sizes of the subtrees of $\mathrm{val}_{\mathcal{G}}(t_B^0)[y/\mathrm{val}_{\mathcal{G}}(C)] = \mathrm{val}_{\mathcal{G}}(A)$ that appear along the path from the node p_y to the root strictly increase when walking towards the root. This means that there are at most $|N_{B,0}|$ many of these subtrees that belong to the set $\{\mathrm{val}_{\mathcal{G}}(X) \mid X \in N_{B,0}\}$. This allows us to split the string $\mathrm{val}(\mathcal{G}_B)$ into polynomially many substrings. For each of these substrings we can compute a small SLCF string grammar. Moreover, for each substring we can carry out essentially the same argument that we sketched above for ordinary NTAs, because all sibling-constraints are known. In the following, we formally define the splitting of the string $\mathrm{val}(\mathcal{G}_B)$.

For a nonterminal $X \in \mathbb{N}_0 \cap N$ of rank 0, let $s(X) = |\mathrm{val}_{\mathcal{G}}(X)|$ be the number of nodes of the generated tree; this number can be computed in polynomial time by Lemma 4. For a terminal symbol $[A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n, f] \in \Sigma$ of the string grammar $\mathcal{G}_B$ let $s([A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n, f]) = s(A_1) + \cdots + s(A_n) + 1$ (the "+1" comes from the symbol f). The mapping $s : \Sigma \to \mathbb{N}$ is extended to Σ^* in the natural way: $s(a_1 \cdots a_n) = s(a_1) + \cdots + s(a_n)$ for $a_1, \ldots, a_n \in \Sigma$. Finally, for a position $0 \leq p \leq |\mathrm{val}(\mathcal{G}_B)| = |p_y|$ let $s(p) = s(C) + s(\mathrm{val}(\mathcal{G}_B)[1 : p])$ (we assume that $w[i, j] = \varepsilon$ for $i > j$ in the following). The value $s(p)$ is the size of a certain subtree of $\mathrm{val}_{\mathcal{G}}(A) = \mathrm{val}_{\mathcal{G}}(B)[y/\mathrm{val}_{\mathcal{G}}(C)]$, namely the subtree that is rooted in the node $p_y[1 : |p_y| - p]$. This is the node reached by going p steps up (towards the root) from the unique occurrence of y in $\mathrm{val}_{\mathcal{G}}(B)(y)$.

Claim 2. Given p in binary notation, we can compute in polynomial time

the value $s(p)$.

First, we construct in polynomial time an SLCF string grammar for the prefix $\mathrm{val}(\mathcal{G}_B)[1:p]$, which is possible by Lemma 1. Then the number $s(\mathrm{val}(\mathcal{G}_B)[1:p])$ (and hence $s(p)$) can be easily computed bottom-up.

Note that $s(i) < s(j)$ for $i < j$. Hence, there exists a list of numbers $0 \leq p_1 < p_2 < \cdots < p_\ell < |p_y|$ such that (i) $\ell \leq |N_{B,0}|$ and (ii) for all $p \in \{0, \ldots, |p_y|\}$, if $s(p) \in \{s(X) \mid X \in N_{B,0}\}$ then $p = p_i$ for some $1 \leq i \leq \ell$.

Claim 3. The list $p_1, \ldots, p_\ell$ can be computed in polynomial time.

Note that for every $X \in N_{B,0}$ we can compute the size $s(X)$ in polynomial time by Lemma 4. Since $s(i) < s(j)$ for $i < j$, we can use binary search (i) to check whether there exists p with $s(X) = s(p)$, and (ii) to compute p if it exists.

Example 14 (continued). Assume that $s(C) = s(A_1) = 2$, $s(A_2) = 7$ and $s(A_3) = 9$. We have $s(0) = 2$, $s(1) = 7$, $s(2) = 31$, $s(3) = 36$, and $s(4) = 60$. Hence, we obtain the list $(p_1, p_2) = (0, 1)$.

The list $(p_1, \ldots, p_\ell)$ defines the splitting of $\mathrm{val}(\mathcal{G}_B)$ mentioned above. More precisely, using Lemma 1 we compute in polynomial time the symbols $a_i = \mathrm{val}(\mathcal{G}_B)[p_i + 1] \in \Sigma$ and SLCF string grammars $\mathcal{G}_0, \ldots, \mathcal{G}_\ell$ in CNF such that $\mathrm{val}(\mathcal{G}_i) = \mathrm{val}(\mathcal{G}_B)[p_i + 2, p_{i+1}]$ for $0 \leq i \leq \ell$ (here, let $p_0 = -1$ and $p_{\ell+1} = |\mathrm{val}(\mathcal{G}_B)|$). Hence, we have

$$\mathrm{val}(\mathcal{G}_B) \;=\; \mathrm{val}(\mathcal{G}_0)\, a_1\, \mathrm{val}(\mathcal{G}_1)\, a_2 \cdots \mathrm{val}(\mathcal{G}_{\ell-1})\, a_\ell\, \mathrm{val}(\mathcal{G}_\ell).$$

Recall that every prefix of $\mathrm{val}(\mathcal{G}_B)$ represents a tree with a unique occurrence of the parameter y (if this prefix is the empty string then the tree is just y). For $0 \leq i \leq \ell$ let $t_i(y)$ be the tree represented by the prefix $\mathrm{val}(\mathcal{G}_0)\, a_1 \cdots \mathrm{val}(\mathcal{G}_{i-1})\, a_i$ (thus $t_0(y) = y$) and let $u_i(y)$ be the tree represented by the prefix $\mathrm{val}(\mathcal{G}_0)\, a_1 \cdots \mathrm{val}(\mathcal{G}_{i-1})\, a_i \mathrm{val}(\mathcal{G}_i)$ (thus $u_\ell(y) = t_B^0(y)$). In our example we have $t_0(y) = u_0(y) = y$, $t_1(y) = u_1(y) = g(A_1, y, A_1)$, $t_2(y) = f(A_2, A_2, t_1(y), A_3)$, and $u_2(y) = t_B^0(y)$.

We compute the state sets $P_i = \widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(t_i[y/C]))$ and $Q_i = \widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(u_i[y/C]))$ successively in polynomial time. We start with $P_0 = \widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(C))$; recall that this set is already computed.

Computing the set P_i from Q_{i-1} ($i > 0$) is straightforward: assume that $a_i = [A_1, \ldots, A_{j-1}, y, A_j, \ldots, A_n, f]$. Hence, we have

$$t_i(y) = f(A_1, \ldots, A_{j-1}, u_{i-1}(y), A_j, \ldots, A_n).$$

From the SLCF string grammars $\mathcal{G}_0, \ldots, \mathcal{G}_\ell$ we can easily compute a linear and monadic SLCF tree grammar for the tree $\mathrm{val}_{\mathcal{G}}(u_{i-1}[y/C])$. Hence, using Proposition 3, we can check in polynomial time, whether the tree $\mathrm{val}_{\mathcal{G}}(u_{i-1}[y/C])$ equals some $\mathrm{val}_{\mathcal{G}}(A_k)$. Using this information, we can compute in polynomial time the set of states P_i from Q_{i-1}.

In order to compute Q_i from P_i, one has to note that when walking down for $|\mathrm{val}(\mathcal{G}_i)|$ steps from the root of $\mathrm{val}_{\mathcal{G}}(u_i[y/C])$ to the unique node labeled y in $u_i(y)$, then the current subtree is never equal to one of the trees rooted in a sibling node (which is a tree $\mathrm{val}_{\mathcal{G}}(X)$ for $X \in N_{B,0}$). Hence, for every terminal symbol $a = [A_1, \ldots, A_{j-1}, y, A_{j+1}, \ldots, A_n, f]$ that occurs in the grammar $\mathcal{G}_i$ we can compute a relation $R_a \subseteq Q \times Q$ as follows (recall that the sets $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(A_k))$ for $k \in \{1, \ldots, n\} \setminus \{j\}$ are already computed):

$$\begin{aligned} R_a = \{(q, q') \in Q \times Q \mid \exists (E, D, q_1, \ldots, q_{j-1}, q, q_{j+1}, \ldots, q_n, f, q') \in \Delta : \\ \forall k \in \{1, \ldots, n\} \setminus \{j\} : q_k \in \widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(A_k)), \\ \forall (k, m) \in E : k = m \vee (k \neq j \neq m \wedge \mathrm{val}_{\mathcal{G}}(A_k) = \mathrm{val}_{\mathcal{G}}(A_m)), \\ \forall (k, m) \in D : k = j \vee m = j \vee (k \neq j \neq m \wedge \mathrm{val}_{\mathcal{G}}(A_k) \neq \mathrm{val}_{\mathcal{G}}(A_m))\}. \end{aligned}$$

Next, for every nonterminal X of the SLCF string grammar $\mathcal{G}_i$ we compute a relation R_X as follows: If $X \to a$ is a production of $\mathcal{G}_i$, then set $R_X = R_a$. If $X \to YZ$ is a production of $\mathcal{G}_i$, then set $R_X = R_Y \circ R_X$. Finally, we set $Q_i = \{q' \in Q \mid \exists q \in P_i : (q, q') \in R_X\}$, where X is the start nonterminal of $\mathcal{G}_i$. □

Clearly, the worst-case complexity, in terms of the input grammar, of the procedure in the proof of Theorem 13 exceeds that of Corollary 12: The procedure for Proposition 3 given in [10] constructs two string grammars which realize depth-first left-to-right traversals of the trees represented by the given SLCF tree grammars, and then checks equality of the strings represented by these two string grammars. The best known algorithm, in terms of asymptotic worst-case complexity, is the one by Lifshits [8] which runs in cubic time with respect to the sum of sizes of the given grammars. Since Proposition 3 is applied $O(|N|)$-times, we already obtain a factor of $|\mathcal{G}'|^4$, where $\mathcal{G}'$ is the linear monadic grammar obtained for the given $\mathcal{G}$ through Theorem 10.

7.2. Tree Walking Automata

Recall that the transformation from TWAs into NTAs is inherently exponential. Thus, the following complexity result is not subsumed by our results for NTAs.

Theorem 15. *The problem of checking* $\mathrm{val}(\mathcal{G}) \in L(\mathcal{W})$ *for a given linear SLCF tree grammar* $\mathcal{G}$ *and a given TWA* $\mathcal{W}$ *can be solved in polynomial time.*

Proof. Let $\mathcal{W} = (Q, F, q_I, \Delta)$ be a TWA over $\mathbb{F}$. Let r be the maximal arity of a symbol in the ranked alphabet $\mathbb{F}$. By Theorem 10, we can assume that $\mathcal{G} = (N, P, S)$ is linear and monadic.[1] Moreover, it is easy to modify $\mathcal{W}$ in such a way that the following holds:

- F consists of a single final state q_f.
- $t \in L(\mathcal{W})$ if and only if there exists a sequence

$$(q_0, u_0) \vdash_{\mathcal{W}} (q_1, u_1) \vdash_{\mathcal{W}} \cdots (q_{n-1}, u_{n-1}) \vdash_{\mathcal{W}} (q_n, u_n)$$

  such that $u_0 = u_n = \varepsilon$ and $q_n = q_f$ (i.e., at the end the TWA has to be back at the root).

For every nonterminal $A \in N \cap \mathbb{N}_1$, we compute $4 \cdot (r+1) \cdot |\mathbb{F}|$ binary relations $R^A_{i,f,a,b} \subseteq Q \times Q$, where $i \in \{\varepsilon, 1, \ldots, r\}$, $f \in \mathbb{F}$, and $a, b \in \{0, 1\}$. The idea is that we consider an occurrence of $\mathrm{val}_{\mathcal{G}}(A)$ in a larger tree. The indexes i and f specify the relevant information of this occurrence: i is the type of the root node of $\mathrm{val}_{\mathcal{G}}(A)$ in the whole tree and f is the root symbol of the tree which is substituted for the unique occurrence of the parameter y_1 in $\mathrm{val}_{\mathcal{G}}(A)$. Now we consider a walk of $\mathcal{W}$ which does not leave the occurrence of $\mathrm{val}_{\mathcal{G}}(A)$ and which starts/ends at the root of $\mathrm{val}_{\mathcal{G}}(A)$ or the unique occurrence of the parameter y_1 in $\mathrm{val}_{\mathcal{G}}(A)$. The indexes a and b specify the entry and exit points of the walk, where 0 refers to the root and 1 refers to the parameter y_1 in $\mathrm{val}_{\mathcal{G}}(A)$. A pair (p, q) belongs to the relation $R^A_{i,f,a,b}$, if the TWA $\mathcal{W}$ can enter $\mathrm{val}_{\mathcal{G}}(A)$ in state p at point a and leave $\mathrm{val}_{\mathcal{G}}(A)$ at point b in state q. Moreover, during this walk, we assume that $\mathrm{val}_{\mathcal{G}}(A)$ is embedded in a larger tree in such a way that the root of $\mathrm{val}_{\mathcal{G}}(A)$ is of type i and the parameter y_1 is replaced by the symbol f.

Similarly, for every $A \in N \cap \mathbb{N}_0$, we compute $r + 1$ binary relations $R^A_i \subseteq Q \times Q$, where $i \in \{\varepsilon, 1, \ldots, r\}$. Here, $(p, q) \in R^A_i$, if $\mathcal{W}$ can enter $\mathrm{val}_{\mathcal{G}}(A)$ at its root in state p and leave $\mathrm{val}_{\mathcal{G}}(A)$ at its root in state q under

[1] Theorem 15 can be also shown without the assumption that $\mathcal{G}$ is monadic. The proof becomes only technically more involved, see [40].

the assumption that the root of $\mathrm{val}_{\mathcal{G}}(A)$ has type i in the whole tree. Hence, to decide whether $\mathrm{val}(\mathcal{G}) \in L(\mathcal{W})$, we have to check whether $(q_0, q_f) \in R^S_\varepsilon$.

Now we argue that these relations can easily be computed bottom-up for all nonterminals using dynamic programming. We assume that all productions of $\mathcal{G}$ have one of the types listed in the beginning of the proof of Theorem 13. We first precompute in polynomial time a table for all nonterminals which contains the following information:

- The root symbol $\lambda_{\mathrm{val}_{\mathcal{G}}(A)}(\varepsilon)$ for every nonterminal $A \in N$.
- The unique number $1 \leq i \leq r$ such that the unique occurrence of the parameter y_1 is the i-th child of its parent node in $\mathrm{val}_{\mathcal{G}}(A)$ for every $A \in N \cap \mathbb{N}_1$.

The different types of productions can be dealt with similarly. Let us consider for instance a production of the form $A(y_1) \to B(C(y_1))$. Let $\lambda_{\mathrm{val}_{\mathcal{G}}(C)}(\varepsilon) = g$ and assume that the parameter y_1 is the k-th child of its parent node in $\mathrm{val}_{\mathcal{G}}(B)$. Then, for instance,

$$R^A_{i,f,0,1} = R^B_{i,g,0,1} \circ (R^B_{i,g,1,1} \cup R^C_{k,f,0,0})^* \circ R^C_{k,f,0,1}.$$

The other relations can be computed similarly.

The complexity of our procedure can be roughly estimated as follows. There are $|N \cap \mathbb{N}_1| \cdot 4 \cdot (r+1) \cdot |\mathbb{F}| \leq O(|\mathcal{G}|^3)$ relations for nonterminals of rank 1 and $|N \cap \mathbb{N}_0| \cdot (r+1) \leq O(|\mathcal{G}|^2)$ relations for nonterminals of rank 0. For each of these relations is computed from previously computed relations using a constant number of unions, compositions, and transitive closures. Computing the transitive closure of a relation on Q takes time $O(|Q|^3) \leq O(|\mathcal{A}|^3)$; this is the dominating part in the computation. Hence, in total $O(|\mathcal{G}|^3 \cdot |\mathcal{A}|^3)$ time suffices. □

It is easy to construct a TWA $\mathcal{B}$, which accepts a tree t over the ranked alphabet $\{\wedge, \vee, 0, 1\}$ if and only if t represents a Boolean expression, which evaluates to true. The TWA $\mathcal{B}$ traverses its input tree in depth-first left-to-right order. Since the circuit value problem is PTIME-complete, it follows that already the following question is PTIME-complete: Given a DAG $\mathcal{G}$, is $\mathrm{val}(\mathcal{G})$ accepted by the fixed TWA $\mathcal{B}$? Hence, the upper bound in Theorem 15 is sharp.

An extension of TWAs are TWAs with pebbles where the pebbles are used obeying a stack discipline, i.e., pebbles $1, \ldots, n$ are placed at nodes and

observed, but only the last pebble can be removed, and only the next free pebble can be placed. It is known that TWAs with pebbles are still less expressive than NTAs, and the transformation from TWAs with pebbles into NTAs is inherently non-elementary [28].

Theorem 16. *The problem of checking for a given TWA with pebbles $\mathcal{W}$ and a linear SLCF tree grammar $\mathcal{G}$, whether* $\mathrm{val}(\mathcal{G}) \in L(\mathcal{W})$ *is* PSPACE*-complete. Moreover,* PSPACE*-hardness already holds for the case that $\mathcal{G}$ is* 0*-bounded (i.e., is a DAG) and $\mathcal{W}$ is deterministic and uses only one pebble.*

Proof. For the upper bound, one can just guess an accepting run of $\mathcal{W}$ on $\mathrm{val}(\mathcal{G})$ incrementally, i.e., at each step we guess and store the next configuration of $\mathcal{W}$. For this, one has to store the current state of $\mathcal{W}$, the current position in $\mathrm{val}(\mathcal{G})$, and the positions of the pebbles. This information can be stored in polynomial space (a node of $\mathrm{val}(\mathcal{G})$ can be represented by a root-leaf path in the unique derivation tree for the tree grammar $\mathcal{G}$, see also [10]). Note that for this argument we do not need the fact that the pebbles are used obeying a stack discipline.

PSPACE-hardness is shown by a simple reduction from quantified Boolean satisfiability (QBF). So, let ψ be a quantified Boolean formula without free variables. Using standard arguments, we can assume that ψ has the form

$$\psi = \forall x_1\, \exists x_2 \cdots \forall x_{2n-1}\, \exists x_{2n} \bigwedge_{i=0}^{m-1} (y_{i,1} \vee y_{i,2} \vee y_{i,3}),$$

where $y_{i,j} \in \{x_1, \neg x_1, \ldots, x_{2n}, \neg x_{2n}\}$. W.l.o.g. we can assume that $m \geq 2$. Our DAG $\mathcal{G}$ generates a kind of binary unfolding of this formula. The productions of $\mathcal{G}$ are:

$$\begin{aligned} A_{2i} &\rightarrow \wedge(A_{2i+1}, A_{2i+1}) \text{ for } 0 \leq i \leq n-1 \\ A_{2i+1} &\rightarrow \vee(A_{2i+2}, A_{2i+2}) \text{ for } 0 \leq i \leq n-1 \\ A_i &\rightarrow \wedge(a, A_{i+1}) \text{ for } 2n \leq i \leq 2n+m-3 \\ A_{2n+m-2} &\rightarrow \wedge(a, a) \end{aligned}$$

Here, a is a constant. Moreover, A_0 is the start nonterminal. Note that the leaves of $\mathrm{val}(\mathcal{G})$ are the nodes $b_1 \cdots b_{2n} 2^i 1$ and $b_1 \cdots b_{2n} 2^{m-1}$, where $b_1, \ldots, b_{2n} \in \{1, 2\}$ and $0 \leq i \leq m-2$. The TWA $\mathcal{W}$ with one pebble works as follows: Basically, $\mathcal{W}$ is the TWA that evaluates Boolean expressions by traversing its input tree in depth-first left-to-right order. When $\mathcal{W}$

visits a leaf $b_1 \cdots b_{2n} 2^i c$ (where $c = 1$ if $0 \leq i \leq m-2$ and $c = \varepsilon$ if $i = m-1$), we can assume that $\mathcal{W}$ has the number i stored in its finite control. Hence, $\mathcal{W}$ knows that the disjunction $y_{i,1} \vee y_{i,2} \vee y_{i,3}$ has to be evaluated in the current truth assignment, which maps the variable x_k to $b_k - 1 \in \{0, 1\}$. Hence, $\mathcal{W}$ has to determine the truth values of the three literals $y_{i,1}$, $y_{i,2}$, $y_{i,3}$. The truth value of a variable x_k ($1 \leq k \leq 2n$) can be easily determined as follows: First $\mathcal{W}$ places the pebble on the current leaf $b_1 \cdots b_{2n} 2^i c$. Then it walks up for exactly $|c| + i + (2n - k)$ steps. If the reached node is a left (resp. right) child of its parent node then x_k evaluates to false (resp. true). Then, $\mathcal{W}$ can deterministically walk back to the leaf, where the pebble was placed before, by making a depth-first left-to-right traversal. □

8. Adding Nondeterminism, Non-Linearity or Recursion

If we relax condition (i) of the definition of SLCF tree grammars to (i') P contains for every $A \in N$ *at least one* production with left-hand side A (but keep the acyclicity condition (ii)) then we obtain *nondeterministic* SLCF tree grammars (NSLCF tree grammars). Such grammars generate finite sets of trees, which by the following example may contain double-exponentially many trees.

8.1. Nondeterminism

Example 17. *For $n \geq 1$, let the linear, productive, and monadic NSLCF tree grammar $\mathcal{G}_n$ consist of the productions*

$$\begin{array}{rcll} S & \to & A_0(a) & \\ A_i(y_1) & \to & A_{i+1}(A_{i+1}(y_1)) & \text{for } 0 \leq i < n \\ A_n(y_1) & \to & f(y_1) & \\ A_n(y_1) & \to & g(y_1). & \end{array}$$

Then $L(\mathcal{G}_n)$ consists of all monadic trees with 2^n many internal nodes, each of which is labeled f or g. Thus $|L(\mathcal{G}_n)| = 2^{2^n}$.

We now want to show that given a linear and productive NSLCF tree grammar $\mathcal{G}$, we can, in general, *not* obtain an equivalent *monadic* grammar of size $|\mathcal{G}|^{O(1)}$. In fact, there is a family $\mathcal{G}_n$ ($n \geq 1$) of linear and productive NSLCF tree grammars such that any monadic, linear, and productive NSLCF tree grammar that generates $L(\mathcal{G}_n)$ is of size $2^{O(|\mathcal{G}_n|^{1/2})}$. Thus, for nondeterministic grammars an exponential blow-up cannot be avoided when going to monadic

grammars. Later we show that this is the worst case blow-up and that in fact any linear and non-deleting NSLCF tree grammar can be transformed into an equivalent monadic one which is at most exponentially larger.

Example 18. *For $n \geq 1$, let the symbol f_n be of rank n and define the linear and productive NSLCF tree grammar $\mathcal{G}_n$ (of size $O(n^2)$) with the following productions:*

$$\begin{aligned} S &\to A_0(a, \ldots, a) \\ A_i(y_1, \ldots, y_n) &\to A_{i+1}(f(y_1), \ldots, f(y_n)) \text{ for } 0 \leq i < n \\ A_i(y_1, \ldots, y_n) &\to A_{i+1}(g(y_1), \ldots, g(y_n)) \text{ for } 0 \leq i < n \\ A_n(y_1, \ldots, y_n) &\to f_n(y_1, \ldots, y_n). \end{aligned}$$

Then $L_n = L(\mathcal{G}_n)$ consists of all trees $f_n(t, t, \ldots, t)$ where t is a monadic tree with n many internal nodes, each of which is labeled f or g.

Lemma 19. *Let $n \geq 1$, $k < n$, and let $\mathcal{G}$ be a linear, non-deleting, and k-bounded NSLCF grammar such that $L(\mathcal{G}) = L_n$ is the set from Example 18. Then $|\mathcal{G}| \geq 2^n$.*

Proof. Assume that $\mathcal{G}$ is a linear, non-deleting, and k-bounded NSLCF tree grammar such that $k < n$ and $L(\mathcal{G}) = L_n$. W.l.o.g. we can assume that every nonterminal of $\mathcal{G}$ appears in a successful derivation of $\mathcal{G}$, i.e., a derivation from the start nonterminal to a terminal tree. Let $P(f_n)$ be the set of all productions of the form $A \to t$, where t contains a subtree of the form $f_n(t_1, \ldots, t_n)$. Clearly, since $\mathcal{G}$ is non-deleting, every right-hand side of a production from $P(f_n)$ contains a unique such subtree. Moreover, in every successful derivation of $\mathcal{G}$, a production from $P(f_n)$ has to be applied exactly once. We claim that $|P(f_n)| \geq 2^n$. Consider a production $(A \to t) \in P(f_n)$ and consider the unique subtree in t of the form $f_n(t_1, \ldots, t_n)$. Since $\mathrm{rank}(A) \leq k < n$ and $\mathcal{G}$ is linear, there exists an $i \in \{1, \ldots, n\}$ such that t_i does not contain a parameter, i.e., $t_i \in T(\mathbb{F} \cup N)$. Assume that two different terminal trees can be derived from t_i. Then we can derive with $\mathcal{G}$ a tree, where the root has two different subtrees, a contradiction. Hence, from t_i we can generate exactly one tree. We denote this tree by $\tau[A \to t]$, since it can be associated with the production $(A \to t) \in P(f_n)$. Hence, for every successful derivation $S \Rightarrow^*_{\mathcal{G}} s$, where the production $(A \to t) \in P(f_n)$ is applied (exactly once), we must have $s = f_n(\tau[A \to t], \ldots, \tau[A \to t])$.

Since we can generate 2^n many terminal trees from S and in each derivation exactly one production from $P(f_n)$ is applied, it follows that $|P(f_n)| \geq 2^n$. □

By the following theorem, the lower bound from Lemma 19 can be matched by an upper bound. The proof of this result is similar to the proof of Theorem 10.

Theorem 20. *For a given linear NSLCF tree grammar $\mathcal{G} = (N, P, S)$ we can construct in time $2^{O(|\mathcal{G}|)}$ a linear and monadic NSLCF tree grammar $\mathcal{G}' = (N', P', S)$ of size $2^{O(|\mathcal{G}|)}$ such that $L(\mathcal{G}') = L(\mathcal{G})$.*

Proof. The proof is very similar to the proof of Theorem 10. Instead of storing just a single skeleton tree sk_A for every nonterminal, we have to store a set SK_A of skeleton trees. The crucial point is that by Lemma 8 the number of different skeleton trees of rank n is bounded exponentially in n. Hence, also the size of the set SK_A is bounded exponentially. For the inductive step in case 2 of the construction of sk_A, we have to combine all trees from SK_B with all trees from SK_C; this yields a set of trees S (instead of the single tree s from (3)). For each tree from S we have to apply contract operations as long as possible in order to obtain SK_A. The formal details are straightforward and left to the reader. □

One might also think about extending Theorem 10 to *non-linear* SLCF tree grammars. But results from [11] make such an extension quite unlikely: it is PSPACE-complete to check whether a deterministic bottom-up tree automaton accepts $\mathrm{val}(\mathcal{G})$, where $\mathcal{G}$ is a given (non-linear) SLCF tree grammar. If we restrict this problem by requiring $\mathcal{G}$ to be k-bounded for a fixed constant k, then it becomes P-complete. Here is an explicit example showing that Theorem 10 cannot be extended to non-linear SLCF tree grammars.

8.2. Non-Linearity

Example 21. *For $n \geq 1$, let the symbol f_n be of rank n, let g have rank 2, and let 0 and 1 have rank 0. Define the productive (but non-linear) SLCF tree grammar $\mathcal{G}_n$ with the following productions, where A_i is a nonterminal of rank i ($1 \leq i \leq n$):*

$$\begin{aligned} S &\to g(A_1(0), A_1(1)) \\ A_i(y_1, \ldots, y_i) &\to g(A_{i+1}(y_1, \ldots, y_i, 0), A_{i+1}(y_1, \ldots, y_i, 1)) \text{ for } 1 \leq i < n \\ A_n(y_1, \ldots, y_n) &\to f_n(y_1, \ldots, y_n) \end{aligned}$$

Then $\mathrm{val}(\mathcal{G}_n)$ *results from a complete binary g-tree of height* n *by replacing the* k*-th leaf (*$0 \leq k \leq 2^n - 1$*) by the tree* $f_n(b_1, \ldots, b_n)$*, where* $b_1 b_2 \cdots b_n$ *is the binary representation of* k*. The size of* $\mathcal{G}_n$ *is* $O(n^2)$*.*

Lemma 22. *Let* $n \geq 1$*,* $k < n$*, and let* $\mathcal{G}$ *be a* k*-bounded SLCF tree grammar such that* $\mathrm{val}(\mathcal{G}) = \mathrm{val}(\mathcal{G}_n)$*, where* $\mathcal{G}_n$ *is the SLCF tree grammar of Example 21. Then* $|\mathcal{G}| \geq 2^{n-k}$*.*

Proof. Let T_n be the set of all occurrences of subterms of the form $f_n(t_1, \ldots, t_n)$ that occur in right-hand sides of $\mathcal{G}$. We claim that $|T_n| \geq 2^{n-k}$. Consider a term $f_n(t_1, \ldots, t_n) \in T_n$. Since $\mathcal{G}$ is k-bounded, at most k parameters can occur among the terms $t_1, \ldots, t_n$. During the derivation, each of these parameters may be either substituted by the constant 0 or 1. Hence, from each $f_n(t_1, \ldots, t_n) \in T_n$, we can obtain during the derivation at most 2^k different trees of the form $f(b_1, \ldots, b_n)$ with $b_1, \ldots, b_n \in \{0, 1\}$. Since $\mathrm{val}(\mathcal{G}_n)$ contains 2^n such subtrees, we get $|T_n| \geq 2^{n-k}$. □

Clearly, Lemma 22 implies that without an exponential blow-up, we cannot reduce the number of parameters in any non-linear SLCF tree grammar to a constant. But we cannot even reduce the number of parameters from n to $\varepsilon \cdot n$ (where $\varepsilon < 1$ is a constant) without an exponential blowup. For arbitrary context-free tree grammars with OI derivation order it is proved in Theorem 6.5 of [41] that the number of parameters gives rise to a hierarchy that is proper at each step (even for the string yield languages).

8.3. Recursion

For arbitrary linear context-free tree grammars (thus, with recursion and nondeterminism), the number of parameters gives rise to a hierarchy of languages which is strict at each level. In fact, the family of languages that can be used to prove the strictness of this hierarchy is similar to the one of Example 18.

Example 23. *For* $n \geq 1$*, let* f_n *be a symbol of rank* n *and* A *be a nonterminal of rank* n*. Define the linear and productive context-free tree grammar* $\mathcal{G}_n$ *with the productions*

$$\begin{aligned} S &\to A(a, \ldots, a) \\ A(y_1, \ldots, y_n) &\to A(f(y_1), \ldots, f(y_n)) \\ A(y_1, \ldots, y_n) &\to f_n(y_1, \ldots, y_n). \end{aligned}$$

Then $L'_n = L(\mathcal{G}_n)$ consists of all trees $f_n(t, t, \ldots, t)$ where t is a monadic tree of the form $f^m(a)$ for some $m \geq 0$.

The proof of the following lemma is similar to the one of Lemma 19.

Lemma 24. *Let $n \geq 1$ and $k < n$. The set L'_n from Example 23 cannot be generated by a linear, non-deleting, and k-bounded context-free tree grammar.*

Proof. Assume there is a linear, non-deleting, and k-bounded context-free tree grammar $\mathcal{G}$ such that $k < n$ and $L(\mathcal{G}) = L'_n$. W.l.o.g. we can assume that every nonterminal of $\mathcal{G}$ appears in a successful derivation of $\mathcal{G}$. Let $P(f_n)$ be the set of all productions of the form $A \to t$, where t contains a subtree of the form $f_n(t_1, \ldots, t_n)$. Since $\mathcal{G}$ is non-deleting, every right-hand side of a production from $P(f_n)$ contains a unique such subtree. Moreover, in every successful derivation of $\mathcal{G}$, a production from $P(f_n)$ has to be applied exactly once.

Consider a production $(A \to t) \in P(f_n)$ and consider the unique subtree in t of the form $f_n(t_1, \ldots, t_n)$. Since $\mathrm{rank}(A) \leq k < n$ and $\mathcal{G}$ is linear, there exists an $i \in \{1, \ldots, n\}$ such that t_i does not contain a parameter, i.e., $t_i \in T(\mathbb{F} \cup N)$. Assume that two different terminal trees can be derived from t_i. Then we can derive with $\mathcal{G}$ a tree, where the root has two different subtrees, a contradiction. Hence, from t_i we can generate exactly one tree that we denote with $\tau[A \to t]$. Thus, for every successful derivation $S \Rightarrow^*_{\mathcal{G}} s$, where the production $(A \to t) \in P(f_n)$ is applied, we must have $s = f_n(\tau[A \to t], \ldots, \tau[A \to t])$. Hence $L(\mathcal{G}) = \{f_n(\tau[A \to t], \ldots, \tau[A \to t]) \mid (A \to t) \in P(f_n)\}$ is finite, a contradiction. □

Let us emphasize that it is crucial for Lemma 19 and 24 that the arity of the root symbol f_n (which is n) is greater than k.

9. Future Work

It will be interesting to investigate the practical implications of our results. For instance, in [17], tree automata over linear SLCF grammars are used for efficient XPath execution. Is it possible to improve running times by first transforming the grammars into monadic grammars? A similar question can be raised concerning other problems such as equivalence checking or unification over SLCF grammars. The last two problems have recently been implemented [18].

As mentioned in the Introduction, tree automata with equality and disequality constraints between sibling nodes (NTACs) have recently been generalized [25]. Other even more powerful recent models are tree automata with arbitrary disequality and restricted equality constraints [42] and tree automata with global constraints [43, 44]. Can we extend our results and give polynomial time algorithms for evaluating such automata over linear SLCF grammars? Another missing point is to determine precise polynomials for Theorems 13 and 15. Moreover, can we prove any lower bounds on automata evaluation over SLCF grammars? An interesting problem is to study restrictions of non-linear and nondeterministic grammars which still allow a polynomial time transformation into monadic (or into k-bounded for a constant k) SLCF grammars.

Acknowledgments The first author is supported by the DFG research project *Algorithms on compressed data* (ALKODA). We would like to thank Christian Mathissen for pointing out a mistake in a previous version of the contract-2 operation.

References

[1] W. Rytter, Grammar compression, LZ-encodings, and string algorithms with implicit input, in: 31st Internat. Colloquium on Automata, Languages and Programming, ICALP'04, volume 3142 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, 2004, pp. 15–27.

[2] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, A. Shelat, The smallest grammar problem., IEEE Transactions on Information Theory 51 (2005) 2554–2576.

[3] R. Carrascosa, F. Coste, M. Gallé, G. G. I. López, Choosing word occurrences for the smallest grammar problem, in: 4th Internat. Conference on Language and Automata Theory and Applications, LATA'10, volume 6031 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, 2010, pp. 154–165.

[4] M. Karpinski, W. Rytter, A. Shinohara, An efficient pattern-matching algorithm for strings with short descriptions, Nord. J. Comput. 4 (1997) 172–186.

[5] M. Miyazaki, A. Shinohara, M. Takeda, An improved pattern matching algorithm for strings in terms of straight-line programs, J. of Discrete Algorithms 1 (2000) 187–204.

[6] W. Rytter, Compressed and fully compressed pattern-matching in one and two dimensions, Proceedings of IEEE 88 (2000) 1769–1778.

[7] W. Plandowski, Testing equivalence of morphisms on context-free languages, in: Second European Symposium on Algorithms, ESA'94, volume 855 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, 1994, pp. 460–470.

[8] Y. Lifshits, Processing compressed texts: A tractability border, in: 18th Annual Symposium on Combinatorial Pattern Matching, CPM'07, volume 4580 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, 2007, pp. 228–240.

[9] M. Schmidt-Schauß, G. Schnitger, Fast equality test for straight-line compressed strings, Frank report 45, Institut für Informatik. Fachbereich Informatik und Mathematik. J. W. Goethe-Universität Frankfurt am Main, 2011.

[10] G. Busatto, M. Lohrey, S. Maneth, Efficient memory representation of XML document trees, Inf. Syst. 33 (2008) 456–474.

[11] M. Lohrey, S. Maneth, The complexity of tree automata and XPath on grammar-compressed trees, Theor. Comput. Sci. 363 (2006) 196–210.

[12] J. Lamping, An algorithm for optimal lambda calculus reductions, in: 17th Annual ACM Symposium on Principles of Programming Languages, POPL'90, ACM Press, 1990, pp. 16–30.

[13] M. Lohrey, S. Maneth, R. Mennicke, Tree structure compression with repair, in: Data Compression Conference, DCC 2011, IEEE Computer Society, 2011, pp. 353–362.

[14] P. Buneman, M. Grohe, C. Koch, Path queries on compressed XML, in: 29th Internat. Conference on Very Large Data Bases, VLDB'03, Morgan Kaufmann, 2003, pp. 141–152.

[15] M. Schmidt-Schauß, Polynomial Equality Testing for Terms with Shared Substructures, Frank report 21, Institut für Informatik. Fachbereich Informatik und Mathematik. J. W. Goethe-Universität Frankfurt am Main, 2005.

[16] D. K. Fisher, S. Maneth, Structural selectivity estimation for XML documents, in: 23rd Internat. Conference on Data Engineering, ICDE'07, IEEE Computer Society Press, 2007, pp. 626–635.

[17] S. Maneth, T. Sebastian, Fast and tiny structural self-indexes for XML, CoRR abs/1012.5696 (2010).

[18] A. Gascón, S. Maneth, L. Ramos, First-order unification on compressed terms, in: 22nd Internat. Conference on Rewriting Techniques and Applications, RTA 2011, volume 10 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011, pp. 51–60.

[19] A. Gascón, G. Godoy, M. Schmidt-Schauß, Unification and matching on compressed terms, ACM Trans. Comput. Log. 12 (2011). Electronic, 26 pages.

[20] A. Gascón, G. Godoy, M. Schmidt-Schauß, Unification with singleton tree grammars, in: 20th Internat. Conference on Rewriting Techniques and Applications, RTA'09, volume 5595 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, 2009, pp. 365–379.

[21] A. Gascón, G. Godoy, M. Schmidt-Schauß, Context matching for compressed terms, in: 23rd Annual IEEE Symposium on Logic in Computer Science, LICS'08, IEEE Computer Society, 2008, pp. 93–102.

[22] J. Levy, M. Schmidt-Schauß, M. Villaret, The complexity of monadic second-order unification, SIAM J. Comput. 38 (2008) 1113–1140.

[23] B. Bogaert, S. Tison, Equality and disequality constraints on direct subterms in tree automata, in: 9th Symposion on Theoretical Aspects of Computer Science, STACS'92, volume 577 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, 1992, pp. 161–172.

[24] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, C. Löding, D. Lugiez, S. Tison, M. Tommasi, Tree automata techniques and applications, Available at: `http://www.grappa.univ-lille3.fr/tata`, 2007.

[25] H. Comon-Lundh, F. Jacquemard, N. Perrin, Tree automata with memory, visibility and structural constraints, in: 10th Internat. Conference on Foundations of Software Science and Computational Structures, FoSSaCS'07, volume 4423 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, 2007, pp. 168–182.

[26] M. Bojanczyk, T. Colcombet, Tree-walking automata do not recognize all regular languages, SIAM J. Comput. 38 (2008) 658–701.

[27] M. Bojanczyk, Tree-walking automata, 2008. Survey presented as tutorial at LATA 2008, available at: `http://www.mimuw.edu.pl/~bojan/papers/twasurvey.pdf`.

[28] M. Samuelides, L. Segoufin, Complexity of pebble tree-walking automata, in: 16th Internat. Symposium on Fundamentals of Computation Theory, FCT'07, volume 4639 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, 2007, pp. 458–469.

[29] M. Lohrey, S. Maneth, M. Schmidt-Schauß, Parameter reduction in grammar-compressed trees, in: 12th Internat. Conference on Foundations of Software Science and Computational Structures, FOSSACS'09, volume 5504 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, 2009, pp. 212–226.

[30] J. W. Hopcroft, J. D. Ullman, Introduction to automata theory, languages, and computation, Addison-Wesley, 1979.

[31] J. E. Hopcroft, R. Motwani, J. D. Ullman, Introduction to automata theory, languages, and computation - international edition (2. ed), Addison-Wesley, 2003.

[32] F. Gécseg, M. Steinby, Tree languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Volume 3, Springer, 1997, pp. 1–68.

[33] B. Courcelle, A representation of trees by languages I, Theoret. Comput. Sci. 6 (1978) 255–279.

[34] A. V. Aho, J. D. Ullman, Translations on a context-free grammar, Inform. and Control 19 (1971) 439–475.

[35] J. Engelfriet, H. J. Hoogeboom, B. Samwel, XML transformation by tree-walking transducers with invisible pebbles, in: 26th Symposium on Principles of Database Systems, PODS'07, ACM Press, 2007, pp. 63–72.

[36] M. Fischer, Grammars with macro-like productions, Ph.D. thesis, Harvard University, Massachusetts, 1968.

[37] A. Fujiyoshi, T. Kasai, Spinal-formed context-free tree grammars, Theory Comput. Syst. 33 (2000) 59–83.

[38] J. Levy, M. Schmidt-Schauß, M. Villaret, Bounded second-order unification is NP-complete, in: 17th International Conference on Term Rewriting and Applications, RTA'06, volume 4098 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, 2006, pp. 400–414.

[39] J. Levy, M. Schmidt-Schauß, M. Villaret, On the complexity of bounded second-order unification and stratified context unification, Logic Journal of the IGPL 19 (2011) 763–789.

[40] M. Saadi, Auswertung von Tree Walking Automaten auf komprimierten Daten, Master's thesis, Universität Leipzig, 2009.

[41] J. Engelfriet, G. Rozenberg, G. Slutzki, Tree transducers, L systems, and two-way machines, J. Comp. Syst. Sci. 20 (1980) 150–202.

[42] G. Godoy, O. Giménez, L. Ramos, C. Àlvarez, The HOM problem is decidable, in: 42nd ACM Symposium on Theory of Computing, STOC'10, ACM Press, 2010, pp. 485–494.

[43] L. Barguñó, C. Creus, G. Godoy, F. Jacquemard, C. Vacher, The emptiness problem for tree automata with global constraints, in: Proc. LICS 2010, IEEE Computer Society Press, 2010, pp. 263–272.

[44] E. Filiot, J.-M. Talbot, S. Tison, Tree automata with global constraints, Int. J. Found. Comput. Sci. 21 (2010) 571–596.