Parameter Reduction in Grammar-Compressed Trees

Markus Lohrey[1], Sebastian Maneth[2], and Manfred Schmidt-Schauß[3]

[1] Universität Leipzig, Institut für Informatik, Germany
[2] NICTA and University of New South Wales, Australia
[3] Johann Wolfgang Goethe-Universität Frankfurt, Institut für Informatik, Germany
lohrey@informatik.uni-leipzig.de, sebastian.maneth@nicta.com.au
schauss@cs.uni-frankfurt.de

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, a polynomial time algorithm is presented for testing whether a given nondeterministic tree automaton with sibling constraints accepts a tree given by a linear straight-line context-free tree grammar. It is 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.

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 [13] 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., [13] for further details.
- 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 the pattern matching problem for compressed strings, see the references in [13].

Due to these appealing properties, it is natural to generalize SLPs to other more complex data structures. For trees, this is done in [3, 11]. In this context, 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 in [3, 11]. They generalize the sharing of repeated subtrees in a tree as it is well-known from DAGs (directed acyclic graphs) to the sharing of repeated subpatterns in a tree (a subpattern is a connected subgraph of the tree). In the context of commonly used XML documents, experiments show that SLCF tree grammars can give approximately 2-3 times higher compression ratios [3] than DAGs [2] when compressing document tree structures. Since sharing of patterns in an SLCF tree grammar can occur along the paths of a tree, it is possible to have a grammar of size $O(n)$[1] that generates a tree of height 2^n; this is not possible with a DAG (the 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 [13], 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. Note that our tree compression algorithm BPLEX [3] generates a small *linear* SLCF tree grammar for a given input tree. It can be checked in polynomial time whether two linear SLCF tree grammars generate the same tree [3, 14]. This result generalizes a corresponding result for (string) SLPs of Plandowski [12]. It remains open whether polynomial time equality testing is also possible for non-linear SLCF tree grammars.

Another useful restriction on SLCF tree grammars is *k-boundedness* (for some fixed k): 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 7). 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 [9, 10, 14], where they are called *singleton tree grammars*. We present two algorithmic applications of Theorem 7: it can be tested in polynomial time whether a given tree automaton accepts the tree given by a linear SLCF tree grammar (Corollary 9). 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 9 to tree automata with equality and disequality constraints between sibling nodes [1, 4]. These are bottom-up tree automata which can test whether the subtrees rooted at children of

[1] The size of a grammar is defined as the sum of the sizes of all right-hand sides of productions.

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 [5]). The running time of this second polynomial time algorithm is much worse than the running time stated in Corollary 9 for ordinary tree automata; therefore we state the two results separately.

In Section 7 we show that Theorem 7 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 7, 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 full version of this paper including all proofs will appear.

2 Trees and SLCF Tree Grammars

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 $\{(v, vi) \in \mathrm{dom}_t \times \mathrm{dom}_t \mid v \in \mathbb{N}^*, i \in \mathbb{N}\}$. The size of t is $|t| = |\mathrm{dom}_t|$. 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 $T(\mathbb{F})$. 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 by the 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 $A(y_1 \ldots, y_n) \to t(y_1, \ldots, y_n)$ 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 $s' = C[t[y_1/t_1 \cdots y_n/t_n]]$. Let $L(\mathcal{G}) = \{t \in T(\mathbb{F}) \mid S \Rightarrow_{\mathcal{G}}^{*} t\} \subseteq T(\mathbb{F})$. 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.
- $\mathcal{G}$ is *non-erasing*, if $t \notin \mathbb{Y}$ for every production $(A \to t) \in P$.
- $\mathcal{G}$ 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 *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, which we denote with $\mathrm{val}(\mathcal{G})$. 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 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 $\mathrm{val}(\mathcal{G}) = \mathrm{val}_{\mathcal{G}}(S)$. SLCF tree grammars generalize *string* generating straight-line programs [13] 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 1. Let the (non-linear) monadic SLCF tree grammar $\mathcal{G}_n$ consist of the productions $S \to A_0(a)$, $A_i(y_1) \to A_{i+1}(A_{i+1}(y_1))$ for $0 \leq i < n$, and $A_n(y_1) \to f(y_1, y_1)$. 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 $\mathcal{G}$ one has $|\mathrm{val}(\mathcal{G})| \leq 2^{O(|\mathcal{G}|)}$. Thus, linear SLCF tree grammars have at most exponential compression ratios, just like DAGs, which can be seen 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 $0 \leq i < n$, 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 [3].

Theorem 2. *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 Theorem 2 can be generalized to non-linear SLCF tree grammars. In [3] we could only prove a PSPACE upper bound for the equality problem for non-linear SLCF tree grammars.

3 Tree Automata

Let $\mathbb{F}$ be a ranked alphabet. 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 $\mathcal{A}$ is $L(\mathcal{A}) = \{t \in T(\mathbb{F}) \mid \widetilde{\Delta}(t) \cap F \neq \emptyset\}$. The *size* of the NTA $\mathcal{A} = (Q, \Delta, F)$ is defined as $|\mathcal{A}| = \sum_{(q_1,\ldots,q_n,f,q)\in\Delta}(n \cdot \log|Q| + \log|\mathbb{F}|)$.

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 $t_1, \ldots, t_n \in T(\mathbb{F})$:

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

Again, 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 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 indices 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 $|\mathcal{G}|$.

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 $\operatorname{rank}(A) = n$, the term t_A has one of the following two forms:

(a) $f(y_1, \dots, y_n)$ with $f \in \mathbb{F}_n$
(b) $B(y_1, \dots, y_{i-1}, C(y_i, \dots, y_{j-1}), y_j, \dots, y_n)$ with $B, C \in N$, $1 \le i \le j \le n+1$.

The proof of the following proposition is a straightforward extension of the corresponding construction for context-free string grammars.

Proposition 3. *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 $N' \supseteq N$, $|N'| \le 2 \cdot |\mathcal{G}|$, $\mathcal{G}'$ is k'-bounded, $k' \le 2r - 1$, and $\operatorname{val}_{\mathcal{G}'}(A) = \operatorname{val}_{\mathcal{G}}(A)$ for all $A \in N$.*

For macro grammars, a normal form similar to CNF exists (called IO standard form in [7, 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 CNF the parameters have to occur in the order $y_1, \dots, y_{\operatorname{rank}(A)}$ in the right-hand side for A). 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. [4] 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 [8].

Example 4. Consider the linear and productive SLCF tree grammar with the two productions $S \to X(X(a,b), X(b,a))$ and $X(y_1, y_2) \to h(i(y_1), i(y_2))$. An equivalent linear SLCF tree grammar in CNF consists of the following productions:

$$\begin{array}{rcl@{\qquad}rcl}
S & \to & X_0(X_1) & X(y_1, y_2) & \to & Y(I(y_1), y_2) \\
X_0(y_1) & \to & X(y_1, X_2) & Y(y_1, y_2) & \to & H(y_1, I(y_2)) \\
X_1 & \to & X_3(A) & A & \to & a \\
X_2 & \to & X_4(B) & B & \to & b \\
X_3(y_1) & \to & X(y_1, B) & I(y_1) & \to & i(y_1) \\
X_4(y_1) & \to & X(y_1, A) & H(y_1, y_2) & \to & h(y_1, y_2).
\end{array}$$

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 $B(y_1, \dots, y_i, C(y_{i+1}, \dots, y_j), y_{j+1}, \dots, y_m)$ (resp. $f(y_1, \dots, y_n)$) we only need to store the triple (B, C, i) (resp. the symbol f) which has size $O(\log k)$ if the grammar is k-bounded. 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 is again the maximal rank of (non)terminals) which needs only space $O(\log(r) \cdot |\mathcal{G}|)$ in triple notation.

5 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 linear 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 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 $\{y_1, \ldots, y_n\}$.

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 1 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 5. *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.*

Let $\mathcal{G} = (N, P, S)$ be a linear SLCF tree grammar. By Proposition 3 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 we define the tree

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

But this tree is not necessarily a skeleton tree; it may locally violate the conditions (a) and (b) on skeleton trees. Hence, we apply a contract-operation to s which yields the

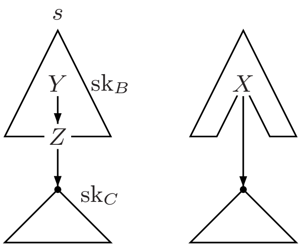


Fig. 1. Contract-1

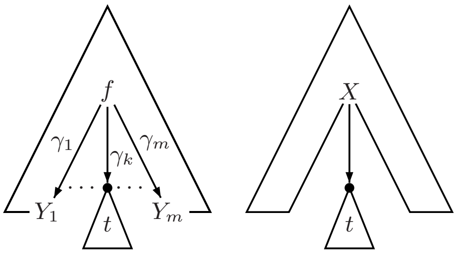


Fig. 2. Contract-2

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

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. 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. In the following consideration, it is useful to set $\varepsilon(t) = t$ for an arbitrary term. Hence, ε is just the identity function on all terms.

Since condition (a) is already satisfied, every subtree t_ℓ ($\ell \neq k$) is of the form $\gamma_\ell(Y_\ell)$ with $Y_\ell \in \mathbb{N}_0$ and $\gamma_\ell \in \{\varepsilon\} \cup \mathbb{N}_1$, whereas t_k can be written as $\gamma_k(t)$, where $\gamma_k \in \{\varepsilon\} \cup \mathbb{N}_1$ and t is a tree that does not start with a non-terminal of rank 1. We now do the following:

1. Add a fresh nonterminal $X \in \mathbb{N}_1$ of rank 1 to N'.
2. Add to P' the production
$$X(y_1) \to f(\gamma_1(Y_1), \ldots, \gamma_{k-1}(Y_{k-1}), \gamma_k(y_1), \gamma_{k+1}(Y_{k+1}), \ldots, \gamma_m(Y_m)).$$
3. Replace the subtree
$$f(\gamma_1(Y_1), \ldots, \gamma_{k-1}(Y_{k-1}), \gamma_k(t), \gamma_{k+1}(Y_{k+1}), \ldots, \gamma_m(Y_m))$$
of s by $X(t)$.

After this operation, another contract-1 operation might be necessary (if the new subtree $X(t)$ is below an $\mathbb{N}_1$-labeled node). The resulting tree is the skeleton tree sk_A.

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 6. *For every nonterminal A of $\mathcal{G}$ we have* $\mathrm{val}_{\mathcal{G}}(A) = \mathrm{val}_{\mathcal{G}'}(\mathrm{sk}_A)$.

Theorem 7. *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 4, 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 (1), 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 5 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 6 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 [10] under the name *singleton tree grammar* (STG). They are, by Theorem 7, polynomially equivalent to finite unions of linear SLCF grammars and hence their results can be applied for linear grammars.

Example 8. We transform the linear CNF grammar constructed in Example 4 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 Y we obtain without contract operations:

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

Let us now construct sk_{X_4}, sk_{X_3}, sk_{X_2}, sk_{X_1}, sk_{X_0}, and sk_S in this order:

- construction of sk_{X_4}: For the tree s in (1) we obtain $s = h(I(y_1), I(A))$. With contract-2, we obtain the new production $C(y_1) \to h(I(y_1), I(A))$ and the skeleton tree $\mathrm{sk}_{X_4} = C(y_1)$.
- Construction of sk_{X_3}: we get $s = h(I(y_1), I(B))$. With contract-2, we obtain the new production $D(y_1) \to h(I(y_1), I(B))$ and the skeleton tree $\mathrm{sk}_{X_3} = D(y_1)$.
- Construction of sk_{X_2}: we get $s = C(B)$. Thus, we do not add a new production to P' and set $\mathrm{sk}_{X_2} = C(B)$.
- Construction of sk_{X_1}: we get $s = D(A)$. Again, we do not add a new production to P' and set $\mathrm{sk}_{X_1} = D(A)$.
- Construction of sk_{X_0}: we get $s = h(I(y_1), I(C(B)))$. A first contract-1 operation adds the production $E(y_1) \to I(C(y_1))$ to P' and updates s to $s = h(I(y_1), E(B))$. Now, we have to apply another contract-2 operation, which adds the production $F(y_1) \to h(I(y_1), E(B))$ to P'. We set $\mathrm{sk}_{X_0} = F(y_1)$.

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

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

$$\begin{array}{lll} S \to G(A) & C(y_1) \to h(I(y_1), I(A)) & F(y_1) \to h(I(y_1), E(B)) \\ A \to a & D(y_1) \to h(I(y_1), I(B)) & G(y_1) \to F(D(y_1)) \\ B \to b & E(y_1) \to I(C(y_1)) & I(y_1) \to i(y_1) \end{array}$$

6 Applications to Tree Automata Evaluation

In [11], we have shown how to 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 the automaton is a deterministic bottom-up tree automaton then time $O(|\mathcal{G}| \cdot |\mathcal{A}| \cdot n^k)$ suffices. Together with Theorem 7 we obtain the following.

Corollary 9. *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 $\mathrm{val}(\mathcal{G}) \in L(\mathcal{A})$.*

We may assume that $r, k \leq |\mathcal{G}|$ in Corollary 9, 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 9 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. In the rest of this section, we extend this result to tree automata with sibling-constraints.

Theorem 10. *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 7 we can assume that $\mathcal{G} = (N, P, S)$ is linear and monadic. Moreover, we can assume 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. We will compute for every $A \in \mathbb{N}_0 \cap N$ the set of states $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(A))$. Consider such 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 Theorem 2, 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 a straight-line context-free *string* grammar $\mathcal{G}_B$ as follows:

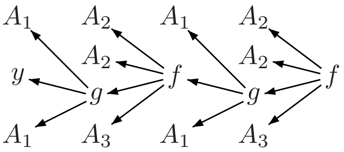
- The set of nonterminals is $\mathbb{N}_1 \cap N$, i.e., the nonterminals of $\mathcal{G}$ of rank 1.
- The set of terminal symbols is $\Sigma = \{[A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n, f] \mid f \in \mathbb{F}_{n+1}, A_1, \ldots, A_n \in \mathbb{N}_0 \cap N, 1 \leq i \leq n+1\}$.
- If $(X(y) \to Y(Z(y))) \in P$, then $\mathcal{G}_B$ contains the production $X \to ZY$; 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 0 is reached.

Example 11. 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 $B \to B_1 B_1$, $B_2 \to [A_2, A_2, y, A_3, f]$, $B_1 \to B_3 B_2$, 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:



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. 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 $s([A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n, f]) = 1 + s(A_1) + \cdots + s(A_n)$. 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 $0 \leq p \leq |\mathrm{val}(\mathcal{G}_B)|$ let $s(p) = s(C) + s(\mathrm{val}(\mathcal{G}_B)[:\, p])$, where $w[:\, k]$ is the prefix of length k of the string w. Also the value $s(p)$ can be computed for a given position p in polynomial time by first constructing in polynomial time an SLCF string grammar for the prefix $\mathrm{val}(\mathcal{G}_B)[:\, p]$. Then the number $s(\mathrm{val}(\mathcal{G}_B)[:\, p])$ can be easily computed bottom-up. 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 obtained by going p steps up (towards the root) from the unique occurrence of y in $\mathrm{val}_{\mathcal{G}}(B)(y)$.

Let us next determine the set $N_{B,0} \subseteq \mathbb{N}_0 \cap N$ of all nonterminals of rank 0 that appear in terminal symbols of $\mathrm{val}(\mathcal{G}_B)$: 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. Let $\{k_1, k_2, \ldots, k_m\} = \{s(X) \mid X \in N_{B,0}\}$, where $k_1 < k_2 < \cdots < k_m$. Also this enumeration can be computed in polynomial time. We now compute a certain splitting of the string $\mathrm{val}(\mathcal{G}_B)$. More precisely, for every $1 \leq i \leq m$ we compute the largest position (i.e. highest position in the tree) $0 \leq p_i \leq |\mathrm{val}(\mathcal{G}_B)|$ such that

$s(p_i) \leq k_i$. This position p_i can be computed in polynomial time with binary search (using the fact that $s(p)$ can be computed in polynomial time for a given p).

Example 11 (continued). Assume that $s(C) = s(A_1) = 2$, $s(A_2) = 7$ and $s(A_3) = 9$. Then, we obtain $k_1 = 2$, $k_2 = 7$, and $k_3 = 9$, as well as $s(0) = 2$, $s(1) = 7$, $s(2) = 31$, $s(3) = 36$, and $s(4) = 60$. Thus, $p_1 = 0$, $p_2 = p_3 = 1$.

From the list $0 \leq p_1 \leq p_2 \leq \cdots \leq p_m \leq |\mathrm{val}(\mathcal{G}_B)|$, we remove every position p_i such that $s(p_i) \neq k_i$ or $p_i = |\mathrm{val}(\mathcal{G}_B)|$. Let $0 \leq p'_1 < p'_2 < \cdots < p'_\ell < |\mathrm{val}(\mathcal{G}_B)|$ be the resulting list. In our example, we only keep $p'_1 = 0$ and $p'_2 = 1$. This list defines our splitting of $\mathrm{val}(\mathcal{G}_B)$. More precisely, we compute in polynomial time the symbols $a_i = \mathrm{val}(\mathcal{G}_B)[p'_i + 1] \in \Sigma$ ($w[p]$ is the p-th symbol of the string w) and SLCF string grammars $\mathcal{G}_0, \ldots, \mathcal{G}_\ell$ such that

$$\mathrm{val}(\mathcal{G}_B) \;=\; \mathrm{val}(\mathcal{G}_0)\, a_1\, \mathrm{val}(\mathcal{G}_1)\, a_2 \cdots \mathrm{val}(\mathcal{G}_{\ell-1})\, a_\ell\, \mathrm{val}(\mathcal{G}_\ell). \tag{2}$$

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 $\mathrm{val}(\mathcal{G}_0)\, a_1 \cdots \mathrm{val}(\mathcal{G}_{i-1})\, a_i \mathrm{val}(\mathcal{G}_i)$. We compute the set of states $P_i = \widetilde{\Delta}(t_i[y/\mathrm{val}_{\mathcal{G}}(C)])$ and $Q_i = \widetilde{\Delta}(u_i[y/\mathrm{val}_{\mathcal{G}}(C)])$ successively. 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 $a_i = [A_1, \ldots, A_{j-1}, y, A_j, \ldots, A_n, f]$. From (2) we can easily compute a monadic SLCF-tree grammar for the tree $u_{i-1}[y/\mathrm{val}_{\mathcal{G}}(C)]$. Hence, using Theorem 2, we can check in polynomial time, whether the tree $u_{i-1}[y/\mathrm{val}_{\mathcal{G}}(C)]$ equals some $\mathrm{val}_{\mathcal{G}}(A_j)$. 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 from the root of $u_i(y)$ to the unique occurrence of y for $|\mathrm{val}(\mathcal{G}_i)|$ steps, then the current subtree is never equal to one of its sibling nodes. 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 transition mapping $\delta_a : Q \to 2^Q$ as follows, where $q \in Q$ (recall that the sets $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(A_k))$ for $k \in \{1, \ldots, n\} \setminus \{j\}$ are already computed):

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

Using the mappings δ_a and the SLCF string grammar $\mathcal{G}_i$, we can compute Q_i from P_i easily in polynomial time. □

7 Adding Nondeterminism or Non-Linearity

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.

Example 12. For $n \geq 1$, let the linear, productive, and monadic NSLCF tree grammar $\mathcal{G}_n$ consist of the productions $S \to A_0(a)$, $A_i(y_1) \to A_{i+1}(A_{i+1}(y_1))$ for $0 \leq i < n$, $A_n(y_1) \to f(y_1)$, and $A_n(y_1) \to g(y_1)$. 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 13. 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 14. *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 13. 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 $\mathcal{G}$. 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 $|P(f_n)| \geq 2^n$. $\square$

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 13.

Example 15. 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 $S \to A(a, \ldots, a)$, $A(y_1, \ldots, y_n) \to A(f(y_1), \ldots, f(y_n))$, and $A(y_1, \ldots, y_n) \to f_n(y_1, \ldots, y_n)$. 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 14.

Lemma 16. *Let $n \geq 1$ and $k < n$. The set L'_n from Example 15 cannot be generated by a linear, non-deleting, and k-bounded context-free tree grammar.*

By the following theorem, the lower bound from Lemma 14 can be matched by an upper bound. The proof of this result is similar to the proof of Theorem 7.

Theorem 17. *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 $L(\mathcal{G}') = L(\mathcal{G})$.*

One might also think about extending Theorem 7 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 7 cannot be extended to non-linear SLCF tree grammars.

Example 18. 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 19. *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 18. 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 19 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 [6] that the number of parameters gives rise to a hierarchy that is proper at each step (even for the string yield languages).

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

References

1. B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In *STACS'92*, LNCS 577, pages 161–171. Springer, 1992.
2. P. Buneman, M. Grohe, and C. Koch. Path queries on compressed XML. In *VLDB 2003*, pages 141–152. Morgan Kaufmann, 2003.
3. G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Information Systems*, 33(4–5):456–474, 2008.
4. H. Comon-Lundh, M. Dauchet, R. Gilleron, F. Jacquemard, C. Löding, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007.
5. H. Comon-Lundh, F. Jacquemard, and N. Perrin. Tree automata with memory, visibility and structural constraints. In *FoSSaCS 2007*, LNCS 4423, pages 168–182, 2007.
6. J. Engelfriet, G. Rozenberg, and G. Slutzki. Tree transducers, L systems, and two-way machines. *J. Comp. Syst. Sci.*, 20:150–202, 1980.
7. M. Fischer. *Grammars with macro-like productions*. PhD thesis, Harvard University, Massachusetts, May 1968.
8. A. Fujiyoshi and T. Kasai. Spinal-formed context-free tree grammars. *Theory Comput. Syst.*, 33(1):59–83, 2000.
9. A. Gascón, G. Godoy, and M. Schmidt-Schauß. Context matching for compressed terms. In *LICS 2008*, pages 93–102. IEEE Computer Society Press, 2008.
10. J. Levy, M. Schmidt-Schauß, and M. Villaret. Bounded second-order unification is NP-complete. In *RTA 2006*, LNCS 4098, pages 400–414, 2006.
11. M. Lohrey and S. Maneth. The complexity of tree automata and XPath on grammar-compressed trees. *Theor. Comput. Sci.*, 363(2):196–210, 2006.
12. W. Plandowski. Testing equivalence of morphisms on context-free languages. In *ESA'94*, LNCS 855, pages 460–470. Springer, 1994.
13. W. Rytter. Grammar compression, LZ-encodings, and string algorithms with implicit input. In *ICALP 2004*, LNCS 3142, pages 15–27. Springer, 2004.
14. M. Schmidt-Schauß. Polynomial equality testing for terms with shared substructures. Technical Report 21, Institut für Informatik, J. W. Goethe-Universität Frankfurt am Main, 2005.