On the Parameterized Complexity of Linear Context-Free Rewriting Systems

Martin Berglund
Umeå University, Sweden
mbe@cs.umu.se

Henrik Björklund
Umeå University, Sweden
henrikb@cs.umu.se

Frank Drewes
Umeå University, Sweden
drewes@cs.umu.se

Abstract

We study the complexity of uniform membership for Linear Context-Free Rewriting Systems, i.e., the problem where we are given a string w and a grammar G and are asked whether $w \in \mathcal{L}(G)$. In particular, we use parameterized complexity theory to investigate how the complexity depends on various parameters. While we focus primarily on rank and fan-out, derivation length is also considered.

1 Introduction

Linear Context-Free Rewriting Systems (LCFRS) were introduced by Vijay-Shanker et al. (1987) with the purpose of capturing the syntax of natural language.[1] It is one of several suggested ways of capturing Joshi's concept of *mildly context-sensitive languages* (Joshi, 1985). As such, it strengthens the expressive power of context-free grammars, while avoiding the full computational complexity of context-sensitive grammars.

One of the defining features of mildly context-sensitive languages is that they should be decidable in polynomial time. This is indeed true for every language that can be generated by an LCFRS. Unlike the case for context-free grammars, however, the *universal* or *uniform* membership problem for LCFRSs, where both the grammar and the string in question are considered as input, is known to be PSPACE-complete (Kaji et al., 1992), making a polynomial time solution very improbable.

The best known algorithms for the problem have a running time of $\mathcal{O}(|G| \cdot |w|^{f \cdot (r+1)})$, where G is the grammar, w is the string, f is the *fan-out* and r is the *rank* of the grammar (Seki et al., 1991; Burden and Ljunglöf, 2005; Boullier, 2004). (For a definition of fan-out and rank, see Section 2.) Unlike the rank of a context-free grammar, the fan-out and rank of an LCFRS cannot in general be reduced to some fixed constant. Increasing the fan-out always gives more generative power, as does increasing the rank while keeping the fan-out fixed (Satta, 1998). The rank can be reduced to two, but at the price of an exponential increase in the fan-out.

Research into algorithms for LCFRS parsing that are efficient enough for practical use is quite active. For example, algorithms for restricted cases are being studied, e.g., by Gómez-Rodríguez et al. (2010), as well as rank reduction, primarily in special cases, where the fan-out is not affected; see, e.g., Sagot and Satta (2010).

This article is a first step towards a finer computational complexity analysis of the membership problem for LCFRSs. Specifically it asks the question "could there exist an algorithm for the uniform LCFRS membership problem whose running time is a fixed polynomial in $|w|$ times an arbitrary function in f and r?" By employing parameterized complexity theory, we show that such an algorithm is very unlikely to be found. Fixing the rank of the grammar to one, the membership problem, parameterized by the fan-out, is W[SAT]-hard. Fixing the fan-out to two and taking the rank as the parameter, the problem is W[1]-hard. Finally, if the fan-out, rank, *and derivation length* are included in the parameter, the problem is W[1]-complete. These results help guide future work, suggesting other types of parameters and grammar restrictions that may yield more favorable complexity results.

2 Preliminaries

For $n \in \mathbb{N}$, we write $[n]$ for $\{1, \ldots, n\}$ and $[n]_0$ for $\{0\} \cup [n]$. Given an alphabet Σ we write Σ^* for all strings over Σ and Σ^+ for all non-empty strings. The empty string is denoted by ε.

[1] Seki et al. (1991) independently suggested the nearly identical Multiple Context-Free Grammars.

Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13), pages 21–29,
Sofia, Bulgaria, August 9, 2013.

2.1 Linear context-free rewriting systems

Let Σ be an alphabet, $x_1, \ldots, x_n$ variables, and $w_1, \ldots, w_k$ strings over Σ such that

$$w_1 \cdots w_k = \alpha_0 \cdot x_{\pi(1)} \cdot \alpha_1 \cdots x_{\pi(n)} \cdot \alpha_n$$

for some permutation π and some strings $\alpha_0, \ldots, \alpha_n \in \Sigma^*$. Then define f as a function over tuples of strings such that

$$f((x_1, \ldots), \ldots, (\ldots, x_n)) = (w_1, \ldots, w_k).$$

A function is *linear regular* if and only if it can be described in this way. For example $f((x_1), (x_2)) = (a, bx_2x_1c)$ is linear regular, and $f((aaa), (bc)) = (a, bbcaaac)$.

Definition 2.1. A *Linear Context-Free Rewriting System* is a tuple $G = (N, \Sigma, F, R, S)$, where N is an alphabet of *nonterminals*, where each $A \in N$ has an associated *fan-out* $\#(A)$; $S \in N$ is the initial nonterminal with $\#(S) = 1$; Σ is an alphabet of *terminals*; F is a set of linear regular functions; and R is a set of rules of the form $A \to g(B_1, \ldots, B_n)$, where $A, B_1, \ldots, B_n \in N$ and g is a function in F of type

$$(\Sigma^*)^{\#(B_1)} \times \cdots \times (\Sigma^*)^{\#(B_n)} \to (\Sigma^*)^{\#(A)}.$$

For rules $A \to g()$, where g has arity 0 and $g() = (\alpha_1, \ldots, \alpha_{\#(A)})$, we often simply write $A \to (\alpha_1, \ldots, \alpha_{\#(A)})$.

The *rank* of a rule is the number of nonterminals on the right-hand side. The rank of G is the maximal rank of any rule in R. The *fan-out* of G is the maximal fan-out of any nonterminal in N.

The language generated by a nonterminal A is a set of n-tuples, where $n = \#(A)$.

Definition 2.2. Let $G = (N, \Sigma, F, R, S)$ be a linear context-free rewriting system. Let $\mathcal{L}_A \subseteq (\Sigma^*)^{\#(A)}$ denote the tuples that a nonterminal $A \in N$ can generate. This is the smallest set such that if $A \to f(B_1, \ldots, B_n)$ is in R then, for all $b_i \in \mathcal{L}_{B_i}$ where $i \in [n]$, $f(b_1, \ldots, b_n) \in \mathcal{L}_A$. The language of G is $\mathcal{L}(G) = \mathcal{L}_S$.

For $i \in \mathbb{N}$, we write i-LCFRS for the class of all LCFRSs of rank at most i and LCFRS(i) for the class of all LCFRSs of fan-out at most i. We also write i-LCFRS(j) for i-LCFRS$\cap$LCFRS(j).

2.2 Parameterized complexity theory

We only reproduce the most central definitions of parameterized complexity theory. For a more thorough treatment, we refer the reader to (Downey and Fellows, 1999; Flum and Grohe, 2006).

A *parameterized problem* is a language $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. The second component is called the *parameter*. An algorithm for $\mathcal{L}$ is *fixed-parameter tractable* if there is a computable function f and a polynomial p such that for every $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm decides in time $f(k) \cdot p(|x|)$ whether $(x, k) \in \mathcal{L}$. The problem of deciding $\mathcal{L}$ is fixed-parameter tractable if there is such an algorithm. If so, $\mathcal{L}$ belongs to the class FPT.

A parameterized problem $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$ is *fpt-reducible* to another parameterized problem $\mathcal{K} \subseteq \Gamma^* \times \mathbb{N}$ if there is a mapping $R : \Sigma^* \times \mathbb{N} \to \Gamma^* \times \mathbb{N}$ such that

1. for all $(x, k) \in \Sigma^* \times \mathbb{N}$, we have $(x, k) \in \mathcal{L}$ if and only if $R(x, k) \in \mathcal{K}$,
2. there is a computable function f and a polynomial p such that $R(x, k)$ can be computed in time $f(k) \cdot p(|x|)$, and
3. there is a computable function g such that for every $(x, k) \in \Sigma^* \times \mathbb{N}$, if $R(x, k) = (y, k')$, then $k' \leq g(k)$.

Note that several parameters may be combined into one by taking their maximum (or sum).

The most commonly used hierarchy of parameterized complexity classes is the following.

$$\text{FPT} \subseteq \text{W}[1] \subseteq \text{W}[2] \subseteq \cdots \subseteq \\ \subseteq \text{W}[\text{SAT}] \subseteq \text{W}[\text{P}] \subseteq \text{XP}$$

The classes W[1],...,W[P] are defined using circuits or, alternatively, logic. None of the inclusions is known to be strict, except that FPT is a strict subclass of XP. It is widely believed, however, that each of them is strict. The class XP is the class of all parameterized problems for which there is a computable function f such that every instance (x, k) can be decided in time $|x|^{f(k)}$.

2.3 Problems of interest

We know from Kaji et al. (1992) that the universal membership problem for 1-LCFRSs is PSPACE-complete. Satta (1992) has further shown that LCFRS(2)-MEMBERSHIP is NP-hard.

We study the following decision problems, where the symbol $\mathfrak{P}$ is used to indicate what the parameter is:

- $\mathfrak{P}$-LCFRS(j)-MEMBERSHIP, where $j \in \mathbb{N}$ is the membership problem for LCFRS(j), parameterized by the rank.

- i-LCFRS($\mathfrak{P}$)-MEMBERSHIP, where $i \in \mathbb{N}$ is the membership problem for i-LCFRS, parameterized by the fan-out.
- $\mathfrak{P}$-LCFRS($\mathfrak{P}$)-MEMBERSHIP is the membership problem for LCFRS parameterized by the rank and the fan-out.
- SHORT $\mathfrak{P}$-LCFRS($\mathfrak{P}$)-DERIVATION is the membership problem for LCFRS parameterized by the rank, the fan-out, and the derivation length.

Since there are algorithms that solve the membership problem for LCFRSs with rank r and fan-out t and string w in time $|w|^{(r+1)t}$ (see, e.g., (Seki et al., 1991; Burden and Ljunglöf, 2005; Boullier, 2004)), we can immediately conclude that $\mathfrak{P}$-LCFRS($\mathfrak{P}$)-MEMBERSHIP, as well as every other parameterized membership problem mentioned above, belongs to XP.

3 Fixed rank grammars

The following theorem establishes a lower bound for 1-LCFRSs parameterized by the fan-out.

Theorem 3.1. 1-LCFRS($\mathfrak{P}$)-MEMBERSHIP *is* W[SAT]*-hard.*

The proof of Theorem 3.1 is by reduction from WEIGHTED MONOTONE SATISFIABILITY. Before we get into the actual proof, we discuss some properties of this problem.

Definition 3.1. A *monotone Boolean formula* is a Boolean formula that contains only conjunctions, disjunctions, and variables. In particular, there are no negations. An instance of WEIGHTED MONOTONE SATISFIABILITY is a pair (ϕ, k), where ϕ is a monotone Boolean formula and $k \in \mathbb{N}$. The question is whether ϕ has a satisfying assignment of weight k, i.e., an assignment that sets exactly k of the variables that occur in ϕ to true. The parameter is k. WEIGHTED MONOTONE SATISFIABILITY is W[SAT]-complete (Abrahamson et al., 1993; Abrahamson et al., 1995; Downey and Fellows, 1999).

We can view a monotone Boolean formula ϕ as an unranked tree, where the root node corresponds to the top level clause and the leaves correspond to bottom level clauses, i.e., variable occurrences. The set $pos(\phi)$ of *positions* of ϕ is defined as usual, consisting of strings of natural numbers that indicate how to navigate to the clauses in a tree representation of ϕ. We denote each subclause of ϕ by C_s, where $s \in pos(\phi)$ is its position. Thus

$$\phi = (((x_1 \wedge (x_2 \vee x_3)) \vee x_3 \vee (x_3 \wedge x_4)) \wedge \\ \wedge x_2 \wedge ((x_1 \wedge (x_2 \vee x_4)) \vee (x_1 \wedge x_3)))$$

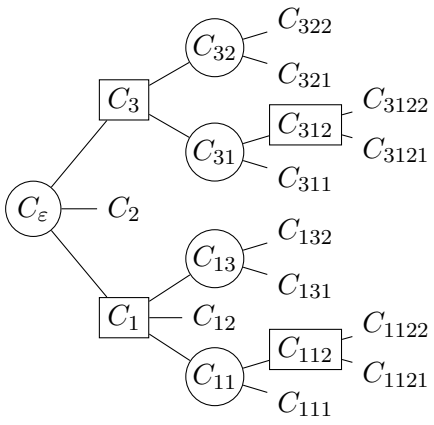


Figure 1: A formula ϕ and its tree representation. Conjunctive clauses are round and disjunctive rectangular. For example, C_{111} is the leftmost occurrence of x_1 and C_{13} the clause $(x_3 \wedge x_4)$.

C_ε denotes the whole of ϕ, while, e.g., C_{ijl} is the lth clause of the jth clause of the ith clause of ϕ. See Figure 1 for an example. We use $\mathcal{C}$ for the set of all clauses of ϕ and $\mathcal{C}_\wedge, \mathcal{C}_\vee$, and $\mathcal{C}_{Var}$ for the sets of conjunctive, disjunctive, and bottom level clauses, respectively. For all $c \in \mathcal{C}_{Var}$ let $Var(c)$ denote the variable in c, and let $Var(\phi)$ denote the set of all variables in ϕ.

Given a monotone Boolean formula ϕ and a variable assignment $\rho\colon Var(\phi) \to \mathbb{B}$, we define a *verification tour* for ϕ and ρ. Such a tour moves through the tree representation of ϕ, starting at the root node, and verifies that ρ satisfies ϕ. To this end, we first define the function $Next : pos(\phi) \to pos(\phi) \cup \{True\}$ as follows. For the root clause let $Next(\varepsilon) = True$. For all $si \in pos(\phi)$, where $s \in \mathbb{N}^*$ and $i \in \mathbb{N}$, if $C_s \in \mathcal{C}_\wedge$ and $s(i+1) \in pos(\phi)$ let $Next(si) = s(i+1)$, otherwise let $Next(si) = Next(s)$.

A verification tour over ϕ, given a variable assignment ρ is constructed by the following procedure. Set the initial position $p = \varepsilon$, then

- If $C_p \in \mathcal{C}_\wedge$ set $p \leftarrow p1$ (i.e., go to the first subclause).
- If $C_p \in \mathcal{C}_\vee$ set $p \leftarrow pi$ for any i (i.e. nondeterministically pick a subclause).
- If $C_p \in \mathcal{C}_{Var}$ verify that $\rho(Var(C_p)) = true$. If so, set $p \leftarrow Next(p)$ and repeat. Otherwise, the verification tour fails.

A verification tour succeeds if it reaches $True$.

The following lemma can be proved by straightforward induction on the structure of ϕ.

Lemma 3.2. *If a verification tour for ϕ and variable assignment ρ succeeds, then ρ satisfies ϕ.*

We are now ready to prove Theorem 3.1.

Proof of Theorem 3.1. Let (ϕ, k) be an instance of WEIGHTED MONOTONE SATISFIABILITY. Let $\{x_1, \ldots, x_n\}$ be the variables that appear in ϕ. In particular, n is the number of distinct variables. Let m be the number of bottom level clauses.

Intuitively, the LCFRS we will construct will guess a weight k variable assignment ρ and then simulate a verification tour for ϕ and ρ.

Basically, we will use one nonterminal per clause and use the structure of the grammar to simulate a verification tour. In order to verify that the necessary bottom level clauses can all be satisfied through the same k true variables, we will use the input string to be parsed. The string w will consist of bracketed sequences of m copies of each of the n variables, i.e., $w = [x_1^m] \cdots [x_n^m]$. To understand the construction of the grammar, please keep in mind that the only derivations that matter are those generating this particular input string.

The grammar will guess which k variables should be set to true and disregard the other variables. Technically, this is done by first letting a nonterminal F generate a tuple of $k+1$ strings $s_0, \ldots, s_k$ such that each s_i consists of zero or more of the bracketed sequences of variables to be disregarded. The rest of the grammar generates exactly k bracketed sequences that will be interleaved with $s_0, \ldots, s_k$. During the generation of these k bracketed sequences it is nondeterministically verified that the corresponding truth assignment satisfies ϕ.

We use the following set of nonterminals:

$$\{S, F\} \cup \{C_s \mid s \in pos(\phi) \cup \{True\}\}$$

For S, there is only one rule: $S \to f_S(F)$. The function f_S places brackets around the k variables that are guessed to be true, represented by the strings $t_1, \ldots, t_k$, and interleaves them with the remaining variables, represented by the strings $s_0, \ldots, s_k$:

$$f_S(s_0, \ldots, s_k, t_1, \ldots, t_k) = (s_0[t_1]s_1 \cdots [t_k]s_k)$$

The nonterminal F has rules $F \to f_{F,i,j}(F)$ for all $i \in [n]$ and $j \in [k]_0$. These rules produce the bracketed sequences of copies of the variables x_i to be disregarded, as can be seen from the corresponding function:

$$f_{F,i,j}(s_0, \ldots, s_k, t_1, \ldots, t_k) = (s_0, \ldots, s_j[x_i^m], \ldots, s_k, t_1, \ldots, t_k)$$

Moreover, there is a single rule

$$F \to f_F(C_\varepsilon)$$

with

$$f_F(t_1, \ldots, t_k) = (\varepsilon, \ldots, \varepsilon, t_1, \ldots, t_k)$$

The rules for the nonterminals that represent clauses differ according to the type of the clause, i.e., if the nonterminal represents a conjunctive clause, a disjunctive clause, or a variable. For each conjunctive clause C_s there is exactly one rule, representing a move to its first subclause. Here, f_{id} is the identity function.

$$C_s \to f_{id}(C_{s1})$$

For every disjunctive clause C_s and every i such that C_{si} is a subclause of C_s there is one rule.

$$C_s \to f_{id}(C_{si})$$

For every bottom level clause, i.e., $C_s \in \mathcal{C}_{Var}$, every $i \in [k]$ and every $j \in [m]$ there is one rule.

$$C_s \to f_{s,i,j}(C_{Next(s)})$$

Intuitively, such a rule corresponds to producing j copies of the variable of clause C_s in component i of the tuple and moving on to the next clause that should be visited in a verification tour. This can be seen from the corresponding function.

$$f_{s,i,j}(t_1, \ldots, t_k) = (t_1, \ldots, Var(C_s)^j t_i, \ldots, t_k)$$

The reason that the function produces j copies of the variable, rather than just one, is that it is unknown beforehand how many times a bottom level clause that represents that particular variable will be visited. Thus the number of copies to be produced has to be guessed nondeterministically in order to make sure that a total of m copies of each variable set to true are eventually produced.

If there is a weight k satisfying assignment, there will also be a verification tour that eventually reaches $True$ when $Next$ is called (by Lemma 3.2). The single rule for C_{True} simply produces a k-tuple of empty strings.

The reduction is polynomial and the fan-out of the resulting grammar is $2k+1$. Thus it is an FPT-reduction. It remains to argue that the grammar can produce w if and only if ϕ has a satisfying assignment of weight k.

We first note that whatever tuple is derived from F, the first $k+1$ entries in the tuple consist of bracketed sequences of the form $[x_l^m]$. If the grammar can produce w, it follows that the tuple $(t_1, \ldots, t_k)$ produced from C_ε must be such that each t_i equals m copies of the same variable name.

Any successful derivation of a string by the grammar corresponds to a verification tour of ϕ and the variable assignment that sets the variables that appear in $(t_1, \ldots, t_k)$ to true and all other variables to false. Thus ϕ has a satisfying assignment of weight k.

For the other direction, assume that ϕ has a satisfying assignment of weight k. Then the grammar can guess this assignment and a corresponding successful verification tour, thus producing w. □

Note that Theorem 3.1 can easily be strengthened to grammars with a binary terminal alphabet. It is enough to represent each variable name by a bitstring of length $\lceil \log_2(m) \rceil$ in the above reduction. We also note that Theorem 3.1 immediately implies that $\mathfrak{P}$-LCFRS($\mathfrak{P}$)-MEMBERSHIP is W[SAT]-hard.

4 Fixed fan-out grammars

We next turn to the case where the fan-out is fixed to two, while the rank is treated as a parameter.

Theorem 4.1. $\mathfrak{P}$-LCFRS(2)-MEMBERSHIP *is* W[1]*-hard.*

Proof. We reduce from k-CLIQUE, the problem of deciding whether a given graph has a clique of size k, with k as the parameter. This problem is known to be W[1]-complete (Flum and Grohe, 2006). Let $G = (\mathsf{V}, \mathsf{E})$ be an undirected graph. We assume, without loss of generality, that $\mathsf{V} = \{1, \ldots, n\}$ and that an edge connecting nodes $i, j \in \mathsf{V}$ is represented as the ordered pair (i, j) such that $i < j$, i.e., $\mathsf{E} \subseteq \{(i, j) \in \mathsf{V} \times \mathsf{V} \mid i < j\}$. To find out whether G has a clique of size k we construct an instance of the membership problem for LCFRSs.

The input alphabet is $\Sigma = \{0, 1\}$. Construct the input string as

$$w = \underbrace{0^n 1 0^n 1 0^n 1 \cdots 1 0^n}_{(3k+2)(k-1)/2 \text{ ones}}.$$

The nonterminals are $N = \{A, E, C, S\}$, with S being the initial nonterminal. The rules are the following.

$\{A \to 0^i \mid i \in \{1, \ldots, n\}\}$.
$\{E \to 0^{n-i} 1 0^{n-j} \mid (i, j) \in \mathsf{E}\}$.
$\{C \to (0^i, 0^{n-i} 1 0^i) \mid i \in \{1, \ldots, n\}\}$.

Handling S is a bit more complex. Let $\phi = k(k-1)/2$, the number of edges in a k-clique. Then the unique rule for S is:

$$S \to f(\underbrace{E, \ldots, E}_{\phi}, \underbrace{C, \ldots, C}_{2\phi}, \underbrace{A, \ldots, A}_{2k}).$$

Now we need to define f. Consider the following application of f.

$$f(e_1, \ldots, e_\phi, (c_1, \hat{c}_1), \ldots, (c_\phi, \hat{c}_\phi), \\ (d_1, \hat{d}_1), \ldots, (d_\phi, \hat{d}_\phi), a_1, \ldots, a_{2k}).$$

The application above evaluates to the string

$$c_1 e_1 d_1 1 c_2 e_2 d_2 1 \cdots \\ \cdots 1 c_\phi e_\phi d_\phi 1 a_1 \theta_1 a_2 1 a_3 \theta_2 a_4 1 s 1 a_{2k-1} \theta_k a_{2k}.$$

The substrings θ_1 through θ_k are left to be defined, and will contain all the $\hat{c}$ and $\hat{d}$ arguments in a careful configuration derived from the structure of a clique. Let $(\pi_1, \pi'_1), \ldots, (\pi_\phi, \pi'_\phi)$ be the lexicographically sorted sequence of edges in a k-clique with nodes numbered 1 through k. For example, $(\pi_1, \pi'_1) = (1, 2)$, $(\pi_2, \pi'_2) = (1, 3)$, $(\pi_k, \pi'_k) = (2, 3)$, and $(\pi_\phi, \pi'_\phi) = (k-1, k)$. Then, for each l, find the longest subsequences $i_1, \ldots, i_p$ and $j_1, \ldots, j_q$ of $1, \ldots, \phi$ for which $\pi_{i_1} = \cdots = \pi_{i_p} = l$ and $\pi'_{j_1} = \cdots = \pi_{j_q} = l$, and let $\theta_l = \hat{c}_{i_1} \cdots \hat{c}_{i_p} \hat{d}_{j_1} \cdots \hat{d}_{j_q}$. □

This construction is simpler than it may at first appear. Basically, the clique is found by generating $k(k-1)/2$ copies of E, each of which will be placed so that it has no choice but to generate an edge in a k-clique. Looking at the first part of the string, each $1 c_l e_l d_l 1$ must generate a string of the form $10^n 10^n 1$: e_l will generate some $0^{n-i} 10^{n-j}$, were (i, j) is an edge in G, which forces c_l to generate 0^i and d_l to generate 0^j. The trick is that c_l and d_l yield the first string in a *pair* generated by an instance of C. The *other* string in the pair describes the same number as the first, but in such a way that it can be carefully placed in the latter part of the derivation string, thus forcing other instances of the C nonterminal to pick *the same*

node (number of zeros) to generate. These are then placed in such a way that the edges picked by the instances of E all belong to the same clique. For example, for $k = 3$ the result of f will be $c_1e_1d_11c_2e_2d_21c_3e_3d_31a_1c_1c_2a_21c_3d_11d_2d_3$, where the latter part ensures that c_1 and c_2 have to pick the same node (lowest-numbered node in the clique), as do c_3 and d_1, and d_2 and d_3.

5 Short derivations

In this section, we consider the length of derivations as an additional parameter. As usual, the length of a derivation is the number of derivation steps it consists of. (In a derivation of an LCFRS (N, Σ, F, R, S), this is the same as the number of applications of functions in F.)

Let $G = (N, \Sigma, F, R, S)$ be an LCFRS in the following. Consider the following problem:

Definition 5.1. An instance of the SHORT $\mathfrak{P}$-LCFRS($\mathfrak{P}$) DERIVATION problem consists of a LCFRS G, some $w \in \Sigma^*$ and a constant $d \in \mathbb{N}$. The question asked is: can w be derived by G in at most d steps? The parameter is $k = d + r + f$ where r is the maximum rank and f the maximum fanout.

Lemma 5.1. SHORT $\mathfrak{P}$-LCFRS($\mathfrak{P}$) DERIVATION *is* W[1]*-hard.*

Proof. The W[1]-hardness of the problem follows immediately from the reduction in the proof of Theorem 4.1, since k-Clique is reduced to an instance of LCFRS membership with $\mathcal{O}(k^2)$ derivation steps, rank $\mathcal{O}(k^2)$, and fixed fan-out. □

We next demonstrate that SHORT $\mathfrak{P}$-LCFRS($\mathfrak{P}$) DERIVATION is *in* W[1] (and is therefore W[1]-complete) by reducing to SHORT CONTEXT-SENSITIVE DERIVATION, shown to be W[1]-complete by Downey et al. (1994). Let $H = (N_H, \Sigma_H, R_H, S_H)$ be an arbitrary context-sensitive grammar in the following. A context-sensitive grammar has nonterminals, terminals and a starting nonterminal just like a LCFRS, but the rules are of the form $\alpha \to \beta$ for strings $\alpha, \beta \in (\Sigma_H \cup N_H)^*$ where $0 < |\alpha| \leq |\beta|$. A derivation starts with the string S_H. A string $w \cdot \alpha \cdot w'$ can be turned into $w \cdot \beta \cdot w'$ in one derivation step if $(\alpha, \beta) \in R_H$.

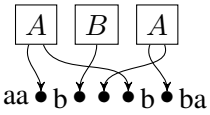
Definition 5.2. An instance of the SHORT CONTEXT-SENSITIVE DERIVATION problem consists of a context-sensitive grammar H, a string $w \in \Sigma_H^*$, and a constant $d_H \in \mathbb{N}$. The question is: can w be derived by H in at most d_H steps? The parameter is d_H.

We are now ready to prove membership in W[1] by a FPT-reduction from (G, w, d) to (H, w, d_H).

Lemma 5.2. *The* SHORT $\mathfrak{P}$-LCFRS($\mathfrak{P}$) DERIVATION *problem is* in W[1].

Proof. We can restrict ourselves to the case where no nonterminal appears twice in a right-hand side of any rule in G. This is because, e.g., a rule of the form $A \to f(B, B)$ can be turned into $A \to f(B, B')$, using a fresh copy B' of B that has the same rules as B (except for having the left-hand side B' rather than B). Note that this modification does not affect the parameter, and increases the size of the grammar only polynomially.

The complete reduction is somewhat lengthy, but the core intuition is very simple. The string is kept the same, and a context-sensitive grammar H is constructed such that $\mathcal{L}(H) = \mathcal{L}(G)$. H simulates G by maintaining a string serialization of the current "configuration" of G, walking through the whole string rewriting the appropriate non-terminal for every rule application in G. A configuration of G can be viewed in this way,



where the derivation has, so far, generated some terminal symbols (the lower-case letters), two instances of the non-terminal A and one instance of B. The configuration keeps track of where the symbols generated by the non-terminals should go in the string, so $\#(A) = 2$, $\#(B) = 1$, and if $(c, d) \in \mathcal{L}_A$ and $e \in \mathcal{L}_B$ this derivation can generate the final string $aacbeddbcba$. These intermediary configurations are in H serialized into strings of nonterminals, with a "nonterminal marker" symbol in each position where a nonterminal is referred to (i.e., H generates a symbol stating "the ith string generated by instance j of the nonterminal A goes here"). H then operates like a Turing machine. A special nonterminal, the rewriting head, picks a rule from G to apply, and walks through the string replacing the nonterminal markers that are affected by that rule. This procedure is then repeated d times.

We start by illustrating the principles of the reduction by an example. Consider the grammar

$\triangleright P_{r_1,1\to2}X_{S,1,1}\triangleleft \Longrightarrow \triangleright X_{A,1,2}X_{A,2,2}P_{r_1,1\to2}\triangleleft \Longrightarrow \triangleright X_{A,1,2}X_{A,2,2}R\triangleleft \Longrightarrow \triangleright X_{A,1,2}RX_{A,2,2}\triangleleft \Longrightarrow$
$\triangleright RX_{A,1,2}X_{A,2,2}\triangleleft \Longrightarrow \triangleright P_{r_2,2\to3}X_{A,1,2}X_{A,2,2}\triangleleft \overset{*}{\Longrightarrow} \triangleright X_{A,1,3}X_{B,1,3}X_{A,2,3}B_{2,3}R\triangleleft \overset{*}{\Longrightarrow}$
$\triangleright P_{r_2,3\to4}X_{A,1,3}X_{B,1,3}X_{A,2,3}X_{B,2,3}\triangleleft \overset{*}{\Longrightarrow} \triangleright X_{A,1,4}X_{B,1,4}X_{B,1,3}X_{A,2,3}X_{B,2,4}X_{B,2,3}R\triangleleft \overset{*}{\Longrightarrow}$
$\triangleright P_{r_6,3\to1}X_{A,1,4}X_{B,1,4}X_{B,1,3}X_{A,2,3}X_{B,2,4}X_{B,2,3}\triangleleft \overset{*}{\Longrightarrow} \triangleright P_{r_3,4\to1}X_{A,1,4}X_{B,1,4}bX_{A,2,3}X_{B,2,4}b\triangleleft \overset{*}{\Longrightarrow}$
$\triangleright P_{r_5,4\to1}aX_{B,1,4}baX_{B,2,4}b\triangleleft \overset{*}{\Longrightarrow} \triangleright aabaabR\triangleleft \overset{*}{\Longrightarrow} aabaab$

Figure 2: A derivation in the context-sensitive grammar constructed to simulate an LCFRS. All steps in the application of the first rule, $r_1 = S \to f(A)$, are given, the rest is abbreviated.

$G = (\{S, A, B\}, \{a, b\}, F, R, S)$ where F is

$$\{f(x, y) = xy, \quad h_a() = (a, a), \quad h_b() = (b, b), \\ g((x, y), (x', y')) = (xx', yy')\},$$

and R contains the following

$$r_1 = S \to f(A) \qquad r_2 = A \to g(A, B)$$
$$r_3 = A \to h_a() \qquad r_4 = A \to h_b()$$
$$r_5 = B \to h_a() \qquad r_6 = B \to h_b()$$

Notice that $\mathcal{L}(G) = \{ww \mid w \in \{a, b\}^+\}$. We now describe how H is constructed by the reduction, after which the more general description follows. A derivation in G starts with the nonterminal S and must then apply r_1. H is constructed to start with the string $\triangleright P_{r_1,1\to2}X_{S,1,1}\triangleleft$ (all these symbols are nonterminals, H has the same terminal alphabet as G). The symbols $\triangleright$ and $\triangleleft$ mark the beginning and end of the string. The nonterminal $X_{S,1,1}$ is a "nonterminal marker" and denotes the location where the *first* string generated by instance 1 of the nonterminal S is to be placed. Since $\#(S) = 1$ the first string is the only string generated from S. The last subscript, the instance number, is there to differentiate markers belonging to different instances of the same nonterminal. The rewriting head non-deterministically picks an instance number for a round of rewriting (single rule application) from a pool sufficiently large to differentiate between the maximal number of nonterminals (since the rank of G is at most k, no more than k^2 nonterminals can be generated in k rule applications). $P_{r_1,1\to2}$ is the "rewriting head", the anchor for rule applications. The subscripts on P determines that it will apply the rule r_1, rewriting nonterminal markers corresponding to the left hand side nonterminal of r_1 which have instance number 1. Applying the rule may create new nonterminal markers, all of which get the instance number 2, also determined by the subscript.

That is, the rules for $P_{r_1,i\to j}$ in H will be $P_{r_1,i\to j}X_{S,1,i} \to X_{A,1,j}X_{A,2,j}P_{r_1,i\to j}$, for $i, j \in [2k^2]$, and $P_{r_1,i\to j}x \to xP_{r_1,i\to j}$ for all other $x \neq \triangleleft$. $P_{r_5,i\to j}X_{B,1,i} \to aP_{r_5,i\to j}$ is another example of a rule corresponding to rule r_5 of G. When a rewriting head hits $\triangleleft$ it is replaced by a nonterminal R which reverses through the string (with rules of the form $xR \to Rx$ for all $x \neq \triangleright$), after which a new rewriting head is nondeterministically picked using one of the rules in $\{\triangleright R \to \triangleright P_{r,i\to j} \mid r \in R, i, j \in [2k^2]\}$, after which the string is rewritten once more. Finally, there are rules $\triangleright \to \varepsilon$, $\triangleleft \to \varepsilon$ and $R \to \varepsilon$, to remove all nonterminals once rewriting has terminated. A derivation is demonstrated in Figure 2.

By induction on the length of derivations, one can show that $\mathcal{L}(H) = \mathcal{L}(G)$. Now we need to modify the construction slightly to ensure that H can simulate d steps of G in d_H steps.

Limiting steps in G. Construct a SHORT $\mathfrak{P}$-LCFRS($\mathfrak{P}$) DERIVATION instance (G', w, d) from (G, w, d) where G' is such that it *cannot* perform more than d derivation steps. Let

$$N' = \{A_i \mid A \in N, i \in [d]\},$$

and let

$$A_i \to f(B_{j_1}, C_{j_2}, \ldots) \in R'$$

for all $A \to f(B, C, \ldots) \in R$, $i \in [d]$ and $j_1 + j_2 + \cdots = i - 1$. Then $G' = (N', \Sigma, R', S_1)$. This reduction is somewhat heavy-handed, but is in FPT since it leaves k unchanged and each rule is replaced by less than k^k rules (since d and the rank of the grammar are part of the parameter k).

Deferring terminals. A problem in completing the reduction from (G, w, d) to (H, w, d_H) is that the number of terminal symbols G generates is not in its parameter k. For example, G may contain a rule like $A \to a \cdots a$, for an arbitrary number of as. Applying this rule may make the intermediary string H is operating on too long for it to complete rewriting in d_H steps. This can

easily be fixed by a polynomial-time rewriting of H. For any rule $w \to w'$ in H such that w' contains at least one terminal, replace every maximal substring $\alpha \in \Sigma^*$ by a new nonterminal T_α, a "terminal place-holder".The rewriting head P and reversal nonterminal R just walk over the place-holders without changing them. Now add the rule $T_\alpha \to \alpha$ for each T_α. For example, where a rewriting head in H might have replaced $X_{A,1,1}$ by $abcX_{B,1,1}baX_{B,2,1}cc$ it will now instead replace it by $T_{abc}X_{B,1,1}T_{ba}X_{B,2,1}T_{cc}$, and can defer replacing the place-holder nonterminals until the end.

Completing the reduction. Now we are ready to put all the pieces together. Given the SHORT $\mathfrak{P}$-LCFRS($\mathfrak{P}$) DERIVATION instance (G, w, d), apply the limiting steps reduction to construct (G', w, d'). Apply the rewriting construction to G to get the context-sensitive grammar H. Now $\mathcal{L}(H)$ equals the language G can generate in d steps. Apply the deferring terminals construction to H to get H'. All that remains is to calculate d_H, the number of steps that H' may take. For an FPT-reduction this number may only depend on the parameter k of (G', w, d'). Picking $d_H = k^5 + 10^3$ is sufficient. Each rule in G' generates less than k nonterminals (since the maximum rank is at most k), each of which will generate at most k markers in the derivation in H' (since the fanout is at most k). The rule may in addition generate $(k+1)k$ terminal place-holders (the k^2 nonterminal markers and string ends separating maximal terminal substrings). After k rule applications, without replacing terminal placeholders, the intermediary string in a derivation in H is less than $k(k^2+(k+1)k)+3$ symbols long. Simulating a rule application in H' entails walking the string twice (forward and then reversing), and k rules are applied, giving $2k(k(k^2+(k+1)k)+3)$ steps. Another $k(k+1)+3$ steps at the end replace the terminal place-holders and remove markers and the rewriting head. Adding things up we arrive at a polynomial of degree 4 that can be rounded up to $k^5 + 10^3$. □

Theorem 5.3. SHORT $\mathfrak{P}$-LCFRS($\mathfrak{P}$) DERIVATION *is* W[1]*-hard.*

Proof. This combines Lemmas 5.1 and 5.2. □

The result of Theorem 5.3 also trivially applies to another natural choice of parameters, the depth of acyclic LCFRS, since they can naturally only take a limited number of derivation steps.

Definition 5.3. A LCFRS is *acyclic* of depth d if d is the smallest integer such that there is a function $\phi : N \to [d]$ such that for all $A \to f(B_1, \ldots, B_n)$ in R and $i \in [n]$ it holds that $\phi(A) < \phi(B_i)$.

Corollary. *The membership problem for acyclic LCFRS where the rank, fan-out, and depth are taken as the parameter is* W[1]*-complete.*

6 Discussion

We have shown that the 1-LCFRS($\mathfrak{P}$)-MEMBERSHIP problem is W[SAT]-hard, but we have no upper bound, except for the trivial XP membership. A conjecture of Pietrzak (2003) may help explain the difficulty of finding such an upper bound. It states that any parameterized problem that has a property that Pietrzak calls *additive* is either in FPT or not in W[P]. Basically, additivity says that any number of instances, sharing a parameter value, can in polynomial time be combined into one big instance, with the same parameter. While 1-LCFRS($\mathfrak{P}$)-MEMBERSHIP is not additive, it has subproblems that are. This means that if Pietrzak's conjecture is true (and FPT $\neq$ W[P]), then 1-LCFRS($\mathfrak{P}$)-MEMBERSHIP cannot belong to W[P].

While our results are mostly intractability results, we see them as a first step towards a more finely grained understanding of the complexity of LCFRS parsing. Ruling out simple parameterization by fan-out or rank as a road towards efficient algorithms lets us focus on other possibilities. Many possible parameterizations remain unexplored. In particular, we conjecture that parameterizing by string length yields FPT membership. In the search for features that can be used in algorithm development, it may also be useful to investigate other formalisms, such as e.g., hypergraph replacement and tree-walking transducers.

Acknowledgments

We acknowledge the support of the Swedish Research Council grant 621-2011-6080.

References

K. A. Abrahamson, R. G. Downey, and M. R. Fellows. 1993. Fixed-parameter intractability II (Extended abstract). In *Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science (STACS'93)*, pages 374–385.

K. A. Abrahamson, R. G. Downey, and M. R. Fellows. 1995. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE-analogues. *Annals of Pure and Applied Logic*, 73:235–276.

P. Boullier. 2004. Range concatenation grammars. In *New Developments in Parsing Technology*, pages 269–289. Kluwer Academic Publishers.

H. Burden and P. Ljunglöf. 2005. Parsing linear context-free rewriting systems. In *Proceedings of 9th International Workshop on Parsing Technologies*.

R. G. Downey and M. R. Fellows. 1999. *Parameterized Complexity*. Springer-Verlag.

R. G. Downey, M. R. Fellows, B. M. Kapron, M. T. Hallett, and H. T. Wareham. 1994. The parameterized complexity of some problems in logic and linguistics. In *Logical Foundations of Computer Science*, pages 89–100. Springer.

J. Flum and M. Grohe. 2006. *Parameterized Complexity Theory*. Springer-Verlag.

C. Gómez-Rodríguez, M. Kuhlmann, and G. Satta. 2010. Efficient parsing of well-nested linear context-free rewriting systems. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL*, pages 276–284.

A. Joshi. 1985. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions. In *Natural Language Parsing*, pages 206–250. Cambridge University Press.

Y. Kaji, R. Nakanisi, H. Seki, and T. Kasami. 1992. The universal recognition problem for multiple context-free grammars and for linear context-free rewriting systems. *IEICE Transactions on Information and Systems*, E75-D(1):78–88.

K. Pietrzak. 2003. A conjecture on the parameterized hierarchy. Notes on a talk given at Dagstuhl Seminar 03311. Unpublished manuscript.

B. Sagot and G. Satta. 2010. Optimal rank reduction for linear context-free rewriting systems with fan-out two. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL'10)*, pages 525–533.

G. Satta. 1992. Recognition of linear context-free rewriting systems. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics (ACL'92)*, pages 89–95.

G. Satta. 1998. Trading independent for synchronized parallelism in finite copying parallel rewriting systems. *J. Computer and System Sciences*, 56(1):27–45.

H. Seki, T. Matsumura, M. Fujii, and T. Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

K. Vijay-Shanker, D. J. Weir, and A. K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Meeting of the Association for Computational Linguists (ACL'87)*, pages 104–111.