Efficient Parsing of Well-Nested Linear Context-Free Rewriting Systems

Carlos Gómez-Rodríguez[1], Marco Kuhlmann[2], and Giorgio Satta[3]

[1]Departamento de Computación, Universidade da Coruña, Spain, cgomezr@udc.es
[2]Department of Linguistics and Philology, Uppsala University, Sweden, marco.kuhlmann@lingfil.uu.se
[3]Department of Information Engineering, University of Padua, Italy, satta@dei.unipd.it

Abstract

The use of well-nested linear context-free rewriting systems has been empirically motivated for modeling of the syntax of languages with discontinuous constituents or relatively free word order. We present a chart-based parsing algorithm that asymptotically improves the known running time upper bound for this class of rewriting systems. Our result is obtained through a linear space construction of a binary normal form for the grammar at hand.

1 Introduction

Since its earliest years, one of the main goals of computational linguistics has been the modeling of natural language syntax by means of formal grammars. Following results by Huybregts (1984) and Shieber (1985), special attention has been given to formalisms that enlarge the generative power of context-free grammars, but still remain below the full generative power of context-sensitive grammars. On this line of investigation, *mildly context-sensitive grammar formalisms* have been introduced (Joshi, 1985), including, among several others, the tree adjoining grammars (TAGs) of Joshi et al. (1975).

Linear context-free rewriting system (LCFRS), introduced by Vijay-Shanker et al. (1987), is a mildly context-sensitive formalism that allows the derivation of tuples of strings, i.e., discontinuous phrases. This feature has been used to model phrase structure treebanks with discontinuous constituents (Maier and Søgaard, 2008), as well as to map non-projective dependency trees into discontinuous phrase structures (Kuhlmann and Satta, 2009).

Informally, in an LCFRS G, each nonterminal can generate string tuples with a fixed number of components. The *fan-out* of G is defined as the maximum number of tuple components generated by G. During a derivation of an LCFRS, tuple components generated by the nonterminals in the right-hand side of a production are concatenated to form new tuples, possibly adding some terminal symbols. The only restriction applying to these generalized concatenation operations is linearity, that is, components cannot be duplicated or deleted.

The freedom in the rearrangement of components has specific consequences in terms of the computational and descriptional complexity of LCFRS. Even for grammars with bounded fan-out, the universal recognition problem is NP-hard (Satta, 1992), and these systems lack Chomsky-like normal forms for fixed fan-out (Rambow and Satta, 1999) that are especially convenient in tabular parsing. This is in contrast with other mildly context-sensitive formalisms, and TAG in particular: TAGs can be parsed in polynomial time both with respect to grammar size and string size, and they can be cast in normal forms having binary derivation trees only.

It has recently been argued that LCFRS might be too powerful for modeling languages with discontinuous constituents or with relatively free word order, and that additional restrictions on the rearrangement of components might be needed. More specifically, analyses of both dependency and constituency treebanks (Kuhlmann and Nivre, 2006; Havelka, 2007; Maier and Lichte, 2009) have shown that rearrangements of argument tuples almost always satisfy the so-called *well-nestedness condition*, a generalization

Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL, pages 276–284,
Los Angeles, California, June 2010.

of the standard condition on balanced brackets. This condition states that any two components x_1, x_2 of some tuple will never be composed with any two components y_1, y_2 of some other tuple in such a way that a 'crossing' configuration is realized.

In this paper, we contribute to a better understanding of the formal properties of well-nested LCFRS. We show that, when fan-out is bounded by any integer $\varphi \geq 1$, these systems can always be transformed, in an efficient way, into a specific normal form with no more than two nonterminals in their productions' right-hand sides. On the basis of this result, we then develop an efficient parsing algorithm for well-nested LCFRS, running in time $\mathcal{O}(\varphi \cdot |G| \cdot |w|^{2\varphi+2})$, where G and w are the input grammar and string, respectively. Well-nested LCFRS with fan-out $\varphi = 2$ are weakly equivalent to TAG, and our complexity result reduces to the well-known upper bound $\mathcal{O}(|G| \cdot |w|^6)$ for this class. For $\varphi > 2$, our upper bound is asymptotically better than the one obtained from existing parsing algorithms for general LCFRS or equivalent formalisms (Seki et al., 1991).

Well-nested LCFRS are generatively equivalent to (among others) coupled context-free grammars (CCFG), introduced by Hotz and Pitsch (1996). These authors also provide a normal form and develop a parsing algorithm for CCFGs. One difference with respect to our result is that the normal form for CCFGs allows more than two nonterminals to appear in the right-hand side of a production, even though no nonterminal may contribute more than two tuple components. Also, the construction in (Hotz and Pitsch, 1996) results in a blow-up of the grammar that is exponential in its fan-out, and the parsing algorithm that is derived runs in time $\mathcal{O}(4^{\varphi} \cdot |G| \cdot |w|^{2\varphi+2})$. Our result is therefore a considerable asymptotic improvement over the CCFG result, both with respect to the normal form construction and the parsing efficiency. Finally, under a practical perspective, our parser is a simple chart-based algorithm, while the algorithm in (Hotz and Pitsch, 1996) involves two passes and is considerably more complex to analyze and to implement than ours.

Kanazawa and Salvati (2010) mention a normal form for well-nested multiple context-free grammars.

Structure In Section 2, we introduce LCFRS and the class of well-nested LCFRS that is the focus of this paper. In Section 3, we discuss the parsing complexity of LCFRS, and show why grammars using our normal form can be parsed efficiently. Section 4 presents the transformation of a well-nested LCFRS into the normal form. Section 5 concludes the paper.

2 Linear Context-Free Rewriting Systems

We write $[n]$ to denote the set of positive integers up to and including n: $[n] = \{1, \ldots, n\}$.

2.1 Linear, non-erasing functions

Let Σ be an alphabet. For integers $m \geq 0$ and $k_1, \ldots, k_m, k \geq 1$, a total function

$$f : (\Sigma^*)^{k_1} \times \cdots \times (\Sigma^*)^{k_m} \to (\Sigma^*)^k$$

is called a *linear, non-erasing function* over Σ with type $k_1 \times \cdots \times k_m \to k$, if it can be defined by an equation of the form

$$f(\langle x_{1,1}, \ldots, x_{1,k_1} \rangle, \ldots, \langle x_{m,1}, \ldots, x_{m,k_m} \rangle) = \vec{\alpha}\,,$$

where $\vec{\alpha}$ is a k-tuple of strings over the variables on the left-hand side of the equation and Σ with the property that each variable occurs in $\vec{\alpha}$ exactly once. The values m and k are called the *rank* and the *fan-out* of f, and denoted by $\rho(f)$ and $\varphi(f)$.

2.2 Linear Context-Free Rewriting Systems

For the purposes of this paper, a *linear context-free rewriting system*, henceforth LCFRS, is a construct $G = (N, T, P, S)$, where N is an alphabet of nonterminal symbols in which each symbol A is associated with a positive integer $\varphi(A)$ called its *fan-out*, T is an alphabet of terminal symbols, $S \in N$ is a distinguished start symbol with $\varphi(S) = 1$; and P is a finite set of productions of the form

$$p \;=\; A \to f(A_1, \ldots, A_m)\,,$$

where $m \geq 0$, $A, A_1, \ldots, A_m \in N$, and f is a linear, non-erasing function over the terminal alphabet T with type $\varphi(A_1) \times \cdots \times \varphi(A_m) \to \varphi(A)$, called the *composition operation* associated with p. The *rank* of G and the *fan-out* of G are defined as the maximal rank and fan-out of the composition operations of G, and are denoted by $\rho(G)$ and $\varphi(G)$.

The sets of *derivation trees* of G are the smallest indexed family of sets D_A, $A \in N$, such that, if

$$p \;=\; A \to f(A_1, \ldots, A_m)$$

$$N = \{S, R\},\ T = \{a, b, c, d\},\ P = \{\, p_1 = S \to f_1(R),\ p_2 = R \to f_2(R),\ p_3 = R \to f_3 \,\},$$
$$\text{where: } f_1(\langle x_{1,1}, x_{1,2}\rangle) = \langle x_{1,1}\, x_{1,2}\rangle,\ f_2(\langle x_{1,1}, x_{1,2}\rangle) = \langle a\, x_{1,1}\, b, c\, x_{1,2}\, d\rangle,\ f_3 = \langle \varepsilon, \varepsilon\rangle .$$

Figure 1: An LCFRS that generates the string language $\{\, a^n b^n c^n d^n \mid n \geq 0 \,\}$.

is a production of G and $t_i \in D_{A_i}$ for all $i \in [m]$, then $t = p(t_1, \ldots, t_m) \in D_A$. By interpreting productions as their associated composition operations in the obvious way, a derivation tree $t \in D_A$ evaluates to a $\varphi(A)$-tuple of strings over T; we denote this tuple by *val*(t). The *string language* generated by G, denoted by $L(G)$, is then defined as

$$L(G) \;=\; \{\, w \in T^* \mid t \in D_S,\ \langle w\rangle = \mathit{val}(t) \,\}.$$

Two LCFRS are called *weakly equivalent*, if they generate the same string language.

Example Figure 1 shows a sample LCFRS G with $\rho(G) = 1$ and $\varphi(G) = 2$. The sets of its derivation trees are $D_R = \{\, p_2^n(p_3) \mid n \geq 0 \,\}$ and $D_S = \{\, p_1(t) \mid t \in D_R \,\}$. The string language generated by G is $\{\, a^n b^n c^n d^n \mid n \geq 0 \,\}$.

2.3 Characteristic strings

In the remainder of this paper, we use the following convenient syntax for tuples of strings. Instead of

$$\langle v_1, \ldots, v_k\rangle, \quad \text{we write} \quad v_1 \,\$\, \cdots \,\$\, v_k,$$

using the \$-symbol to mark the component boundaries. We call this the *characteristic string* of the tuple, and an occurrence of the symbol \$ a *gap marker*. We also use this notation for composition operations. For example, the characteristic string of the operation

$$f(\langle x_{1,1}, x_{1,2}\rangle, \langle x_{2,1}\rangle) \;=\; \langle a\, x_{1,1}\, x_{2,1}, x_{1,2}\, b\rangle$$

is $a\, x_{1,1}\, x_{2,1}\, \$\, x_{1,2}\, b$. If we assume the variables on the left-hand side of an equation to be named according to the schema used in Section 2.1, then the characteristic string of a composition operation determines that operation completely. We will therefore freely identify the two, and write productions as

$$p \;=\; A \to [v_1 \,\$\, \cdots \,\$\, v_k](A_1, \ldots, A_m),$$

where the string inside the brackets is the characteristic string of some composition operation. The substrings $v_1, \ldots, v_k$ are called the *components* of the characteristic string. Note that the characteristic string of a composition operation with type $k_1 \times \cdots \times k_m \to k$ is a sequence of terminal symbols, gap markers, and variables from the set $\{\, x_{i,j} \mid i \in [m],\ j \in [k_i] \,\}$ in which the number of gap markers is $k-1$, and each variable occurs exactly once. When in the context of such a composition operation we refer to 'a variable of the form $x_{i,j}$', then it will always be the case that $i \in [m]$ and $j \in [k_i]$.

The identification of composition operations and their characteristic strings allows us to construct new operations by string manipulations: if, for example, we delete some variables from a characteristic string, then the resulting string still defines a composition operation (after a suitable renaming of the remaining variables, which we leave implicit).

2.4 Canonical LCFRS

To simplify our presentation, we will assume that LCFRS are given in a certain canonical form. Intuitively, this canonical form requires the variables in the characteristic string of a composition operation to be ordered in a certain way.

Formally, the defining equation of a composition operation f with type $k_1 \times \cdots \times k_m \to k$ is called *canonical*, if (i) the sequence obtained from f by reading variables of the form $x_{i,1}$ from left to right has the form $x_{1,1} \cdots x_{m,1}$; and (ii) for each $i \in [m]$, the sequence obtained from f by reading variables of the form $x_{i,j}$ from left to right has the form $x_{i,1} \cdots x_{i,k_i}$. An LCFRS is called *canonical*, if each of its composition operations is canonical.

We omit the proof that every LCFRS can be transformed into a weakly equivalent canonical LCFRS. However, we point out that both the normal form and the parsing algorithm that we present in this paper can be applied also to general LCFRS. This is in contrast to some left-to-right parsers in the literature on LCFRS and equivalent formalisms (de la Clergerie, 2002; Kallmeyer and Maier, 2009), which actually depend on productions in canonical form.

2.5 Well-nested LCFRS

We now characterize the class of *well-nested LCFRS* that are the focus of this paper. Well-nestedness was first studied in the context of dependency grammars (Kuhlmann and Möhl, 2007). Kanazawa (2009)

defines well-nested multiple context-free grammars, which are weakly equivalent to well-nested LCFRS.

A composition operation is called *well-nested*, if it does not contain a substring of the form

$$x_{i,i_1} \cdots x_{j,j_1} \cdots x_{i,i_2} \cdots x_{j,j_2}\,, \quad \text{where } i \neq j\,.$$

For example, the operation $x_{1,1}\, x_{2,1} \$ x_{2,2}\, x_{1,2}$ is well-nested, while $x_{1,1}\, x_{2,1}\, \$\, x_{1,2}\, x_{2,2}$ is not. An LCFRS is called well-nested, if it contains only well-nested composition operations.

The class of languages generated by well-nested LCFRS is properly included in the class of languages generated by general LCFRS; see Kanazawa and Salvati (2010) for further discussion.

3 Parsing LCFRS

We now discuss the parsing complexity of LCFRS, and motivate our interest in a normal form for well-nested LCFRS.

3.1 General parsing schema

A bottom-up, chart-based parsing algorithm for the class of (not necessarily well-nested) LCFRS can be defined by using the formalism of parsing schemata (Sikkel, 1997). The parsing schemata approach considers parsing as a deduction process (as in Shieber et al. (1995)), generating intermediate results called *items*. Starting with an initial set of items obtained from each input sentence, a parsing schema defines a set of *deduction steps* that can be used to infer new items from existing ones. Each item contains information about the sentence's structure, and a successful parsing process will produce at least one *final item* containing a full parse for the input.

The item set used by our bottom-up algorithm to parse an input string $w = a_1 \cdots a_n$ with an LCFRS $G = (N, T, P, S)$ will be

$$\mathcal{I} = \{[A, (l_1, r_1), \ldots, (l_k, r_k)] \mid A \in N \quad \wedge \\ 0 \leq l_i \leq r_i \leq n \quad \forall i \in [k]\},$$

where an item $[A, (l_1, r_1), \ldots, (l_k, r_k)]$ can be interpreted as the set of those derivation trees $t \in D_A$ of G for which

$$\mathit{val}(t) \;=\; a_{l_1+1} \cdots a_{r_1} \,\$\, \cdots \,\$\, a_{l_k+1} \cdots a_{r_k}\,.$$

The set of final items is thus $\mathcal{F} = \{[S, (0, n)]\}$, containing full derivation trees that evaluate to w.

For simplicity of definition of the sets of initial items and deduction steps, let us assume that productions of rank > 0 in our grammar do not contain terminal symbols in their right-hand sides. This can be easily achieved from a starting grammar by creating a nonterminal A_a for each terminal $a \in T$, a corresponding rank-0 production $p_a = A_a \to [a]()$, and then changing each occurrence of a in the characteristic string of a production to the single variable associated with the fan-out 1 nonterminal A_a. With this, our initial item set for a string $a_1 \cdots a_n$ will be

$$\mathcal{H} = \{[A_{a_i}, (i-1, i)] \mid i \in [n]\}\,,$$

and each production $p = A_0 \to f(A_1, \ldots, A_m)$ of G (excluding the ones we created for the terminals) will produce a deduction step of the form given in Figure 2a, where the indexes are subject to the following constraints, imposed by the semantics of f.

1. If the kth component of the characteristic string of f starts with $x_{i,j}$, then $l_{0,k} = l_{i,j}$.
2. If the kth component of the characteristic string of f ends with $x_{i,j}$, then $r_{0,k} = r_{i,j}$.
3. If $x_{i,j} x_{i',j'}$ is an infix of the characteristic string of f, then $r_{i,j} = l_{i',j'}$.
4. If the kth component of the characteristic string of f is the empty string, then $l_{0,k} = r_{0,k}$.

3.2 General complexity

The time complexity of parsing LCFRS with respect to the length of the input can be analyzed by counting the maximum number of indexes that can appear in an instance of the inference rule above. Although the total number of indexes is $\sum_{i=0}^{m} 2 \cdot \varphi(A_i)$, some of these indexes are equated by the constraints.

To count the number of independent indexes, consider all the indexes of the form $l_{0,i}$ (corresponding to the left endpoints of each component of the characteristic string of f) and those of the form $r_{j,k}$ for $j > 0$ (corresponding to the right endpoints of each variable in the characteristic string). By the constraints above, these indexes are mutually independent, and it is easy to check that any other index is equated to one of these: indexes $r_{0,i}$ are equated to the index $r_{j,k}$ corresponding to the last variable $x_{j,k}$ of the ith component of the characteristic string, or to $l_{0,i}$ if there is no such variable; while indexes $l_{j,k}$ with $j > 0$ are equated to an index $l_{0,i}$ if the variable $x_{j,k}$ is at the beginning of a component of the characteristic string, or to an index $r_{j',k'}\,(j' > 1)$ if the variable $x_{j,k}$ follows another variable $x_{j',k'}$.

$$\frac{[A_1, (l_{1,1}, r_{1,1}), \ldots, (l_{1,\varphi(A_1)}, r_{1,\varphi(A_1)})] \quad \cdots \quad [A_m, (l_{m,1}, r_{m,1}), \ldots, (l_{m,\varphi(A_m)}, r_{m,\varphi(A_m)})]}{[A_0, (l_{0,1}, r_{0,1}), \ldots, (l_{0,\varphi(A_0)}, r_{0,\varphi(A_0)})]}$$

(a) The general rule for a parsing schema for LCFRS

$$\frac{[B, (l_1, r_1), \ldots, (l_m, r_m)] \quad [C, (l'_1, r'_1), \ldots (l'_n, r'_n)]}{[A, (l_1, r_1), \ldots, (l_m, r'_1), \ldots (l'_n, r'_n)]} \; r_m = l'_1$$

(b) Deduction step for concatenation

$$\frac{[B, (l_1, r_1), \ldots, (l_m, r_m)] \quad [C, (l'_1, r'_1), \ldots (l'_n, r'_n)]}{[A, (l_1, r_1), \ldots, (l_i, r'_1), \ldots (l'_n, r_{i+1}), \ldots, (l_m, r_m)]} \; r_i = l'_1, r'_n = l_{i+1}$$

(c) Deduction step for wrapping

Figure 2: Deduction steps for parsing LCFRS.

Thus, the *parsing complexity* (Gildea, 2010) of a production $p = A_0 \rightarrow f(A_1, \ldots, A_m)$ is determined by $\varphi(A_0)$ l-indexes and $\sum_{i \in [m]} \varphi(A_i)$ r-indexes, for a total complexity of

$$\mathcal{O}(|w|^{\varphi(A_0) + \sum_{i \in [m]} \varphi(A_i)})$$

where $|w|$ is the length of the input string. The parsing complexity of an LCFRS will correspond to the maximum parsing complexity among its productions. Note that this general complexity matches the result given by Seki et al. (1991).

In an LCFRS of rank ρ and fan-out φ, the maximum possible parsing complexity is $\mathcal{O}(|w|^{\varphi(\rho+1)})$, obtained by applying the above expression to a production of rank ρ and where each nonterminal has fan-out φ. The asymptotic time complexity of LCFRS parsing is therefore exponential both in its rank and its fan-out. This means that it is interesting to transform LCFRS into equivalent forms that reduce their rank while preserving the fan-out. For sets of LCFRS that can be transformed into a *binary* form (i.e., such that all its rules have rank at most 2), the ρ factor in the complexity is reduced to a constant, and complexity is improved to $\mathcal{O}(|w|^{3\varphi})$ (see Gómez-Rodríguez et al. (2009) for further discussion). Unfortunately, it is known by previous results (Rambow and Satta, 1999) that it is not always possible to convert an LCFRS into such a binary form without increasing the fan-out. However, we will show that it is always possible to build such a binarization for *well-nested* LCFRS. Combining this result with the inference rule and complexity analysis given above, we would obtain a parser for well-nested LCFRS running in $\mathcal{O}(|w|^{3\varphi})$ time. But the construction of our binary normal form additionally restricts binary composition operations in the binarized LCFRS to be of two specific forms, *concatenation* and *wrapping*, which further improves the parsing complexity to $\mathcal{O}(|w|^{2\varphi+2})$, as we will see below.

3.3 Concatenation and wrapping

A composition operation is called a *concatenation operation*, if its characteristic string has the form

$$x_{1,1} \; \$ \cdots \$ \; x_{1,m} \, x_{2,1} \; \$ \cdots \$ \; x_{2,n} \, ,$$

where $m, n \geq 1$. Intuitively, such an operation corresponds to the bottom-up combination of two adjacent discontinuous constituents into one. An example of a concatenation operation is the binary parsing rule used by the standard CKY parser for context-free grammars, which combines continuous constituents (represented as 1-tuples of strings in the LCFRS notation). In the general case, a concatenation operation will take an m-tuple and an n-tuple and return an $(m + n - 1)$-tuple, as the joined constituents may have gaps that will also appear in the resulting tuple.

If we apply the general parsing rule given in Figure 2a to a production $A \rightarrow \mathit{conc}(B, C)$, where *conc* is a concatenation operation, then we obtain the deduction step given in Figure 2b. This step uses $2m$ different l- and r-indexes, and $2n - 1$ different l'- and r'-indexes (excluding l'_1 which must equal r_m), for a total of $2m+2n-1 = 2(m+n-1)+1$ indexes. Since $m + n - 1$ is the fan-out of the nonterminal A, we conclude that the maximum number of indexes in the step associated with a concatenation operation in an LCFRS of fan-out φ is $2\varphi + 1$.

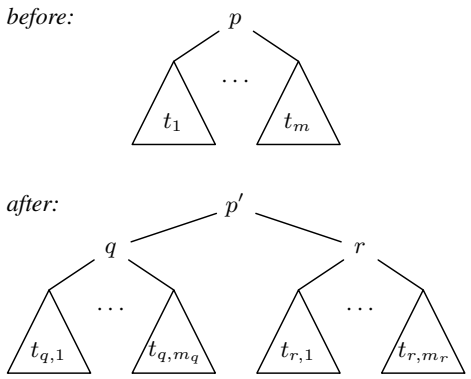

Figure 3: Transformation of derivation trees

A linear, non-erasing function is called a *wrapping operation*, if its characteristic string has the form

$$x_{1,1}\ \$ \cdots \$\ x_{1,i}\, x_{2,1}\ \$ \cdots \$\ x_{2,n}\, x_{1,i+1}\ \$ \cdots \$\ x_{1,m}\,,$$

where $m, n \geq 1$ and $i \in [m-1]$. Intuitively, such an operation wraps the tuple derived from a nonterminal B around the tuple derived from a nonterminal C, filling the ith gap in the former. An example of a wrapping operation is the adjunction of an auxiliary tree in tree-adjoining grammar. In the general case, a wrapping operation will take an m-tuple and an n-tuple and return an $(m+n-2)$-tuple of strings: the gaps of the argument tuples appear in the obtained tuple, except for one gap in the tuple derived from B which is filled by the tuple derived from C.

By applying the general parsing rule in Figure 2a to a production $A \to \mathit{wrap}_i(B, C)$, where wrap_i is a wrapping operation, then we obtain the deduction step given in Figure 2c. This step uses $2m$ different l- and r-indexes, and $2n-2$ different l'- and r'-indexes (discounting l'_1 and r'_n which are equal to other indexes), for a total of $2m+2n-2 = 2(m+n-2)+2$ indexes. Since the fan-out of A is $m+n-2$, this means that a wrapping operation needs at most $2\varphi+2$ indexes for an LCFRS of fan-out φ.

From this, we conclude that an LCFRS of fan-out φ in which all composition operations are either concatenation operations, wrapping operations, or operations of rank 0 or 1, can be parsed in time $\mathcal{O}(|w|^{2\varphi+2})$. In particular, nullary and unary composition operations do not affect this worst-case complexity, since their associated deduction steps can never have more than 2φ indexes.

4 Transformation

We now show how to transform a well-nested LCFRS into the normal form that we have just described.

4.1 Informal overview

Consider a production $p = A \to f(A_1, \ldots, A_m)$, where $m \geq 2$ and f is neither a concatenation nor a wrapping operation. We will construct new productions p', q, r such that every derivation that uses p can be rewritten into a derivation that uses the new productions, and the new productions do not license any other derivations. Formally, this can be understood as implementing a *tree transformation*, where the input trees are derivations of the original grammar, and the output trees are derivations of the new grammar. The situation is illustrated in Figure 3. The tree on top represents a derivation in the original grammar; this derivation starts with the rewriting of the nonterminal A using the production p, and continues with the subderivations $t_1, \ldots, t_m$. The tree at the bottom represents a derivation in the transformed grammar. This derivation starts with the rewriting of A using the new production p', and continues with two independent subderivations that start with the new productions q and r, respectively. The sub-derivations $t_1, \ldots, t_m$ have been partitioned into two sequences

$$t_{1,1}, \ldots, t_{1,m_1} \quad \text{and} \quad t_{2,1}, \ldots, t_{2,m_2}\,.$$

The new production p' will be either a concatenation or a wrapping operation, and the rank of both q and r will be strictly smaller than the rank of p. The transformation will continue with q and r, unless these have rank one. By applying this strategy exhaustively, we will thus eventually end up with a grammar that only has productions with rank at most 2, and in which all productions with rank 2 are either concatenation or wrapping operations.

4.2 Constructing the composition operations

To transform the production p, we first factorize the composition operation f associated with p into three new composition operations f', g, h as follows. Recall that we represent composition operations by their characteristic strings.

In the following, we will assume that no characteristic string starts or ends with a gap marker, or contains immediate repetitions of gap markers. This

property can be ensured, without affecting the asymptotic complexity, by adding intermediate steps to the transformation that we report here; we omit the details due to space reasons. When this property holds, we are left with the following two cases. Let us call a sequence of variables *joint*, if it contains all and only variables associated with a given nonterminal.

Case 1 $f = x_1\, f_1\, x_2 \cdots x_{k-1}\, f_{k-1}\, x_k\, f^*$, where $k \geq 1$, $x_1, \ldots, x_k$ are joint variables, and the suffix f^* contains at least one variable. Let

$$g = x_1\, f_1\, x_2 \cdots x_{k-1}\, f_{k-1}\, x_k\,,$$

let $h = f^*$, and let $f' = \mathit{conc}$. As f is well-nested, both g and h define well-nested composition operations. By the specific segmentation of f, the ranks of these operations are strictly smaller than the rank of f. Furthermore, we have $\varphi(f) = \varphi(g) + \varphi(h) - 1$.

Case 2 $f = x_1\, f_1\, x_2 \cdots x_{k-1}\, f_{k-1}\, x_k$, where $k \geq 2$, $x_1, \ldots, x_k$ are joint variables, and there exist at least one i such that the sequence f_i contains at least one variable. Choose an index j as follows: if there is at least one i such that f_i contains at least one variable and one gap marker, let j be the minimal such i; otherwise, let j be the minimal i such that f_i contains at least one variable. Now, let

$$g = x_1\, f_1\, x_2 \cdots x_j\, \$\, x_{j+1} \cdots x_{k-1}\, f_{k-1}\, x_k\,,$$

let $h = f_j$, and let $f' = \mathit{wrap}_j$. As in Case 1, both g and h define well-nested composition operations whose ranks are strictly smaller than the rank of f. Furthermore, we have $\varphi(f) = \varphi(g) + \varphi(h) - 2$.

Note that at most one of the two cases can apply to f. Furthermore, since f is well-nested, it is also true that at least one of the two cases applies. This is so because for two distinct nonterminals $A_i, A_{i'}$, either all variables associated with $A_{i'}$ precede the leftmost variable associated with A_i, succeed the rightmost variable associated with A_i, or are placed between two variables associated with A_i without another variable associated with A_i intervening. (Here, we have left out the symmetric cases.)

4.3 Constructing the new productions

Based on the composition operations, we now construct three new productions p', q, r as follows. Let B and C be two fresh nonterminals with $\varphi(B) = \varphi(g)$ and $\varphi(C) = \varphi(h)$, and let $p' = A \to f'(B, C)$. The production p' rewrites A into B and C and combines the two subderivations that originate at these nonterminals using either a concatenation or a wrapping operation. Now, let $A_{q,1}, \ldots, A_{q,m_q}$ and $A_{r,1}, \ldots, A_{r,m_r}$ be the sequences of nonterminals that are obtained from the sequence $A_1, \ldots, A_m$ by deleting those nonterminals that are not associated with any variable in g or h, respectively. Then, let

$$q = B \to g(A_{q,1}, \ldots, A_{q,m_q}) \quad \text{and}$$
$$r = C \to h('A_{r,1}, \ldots, A_{r,m_r})\,.$$

4.4 Example

We now illustrate the transformation using the concrete production $p = A \to f(A_1, A_2, A_3)$, where

$$f = x_{1,1}\, x_{2,1}\, \$\, x_{1,2}\, \$\, x_{3,1}\,.$$

Note that this operation has rank 3 and fan-out 3.

The composition operations are constructed as follows. The operation f matches the pattern of Case 1, and hence induces the operations

$$g_1 = x_{1,1}\, x_{2,1}\, \$\, x_{1,2}\,, \quad h_1 = \$\, x_{3,1}\,, \quad f'_1 = \mathit{conc}\,.$$

The productions constructed from these are

$$p'_1 = A \to \mathit{conc}(B_1, C_1)\,,$$
$$q_1 = B_1 \to g_1(A_1, A_2)\,, \quad r_1 = C_1 \to h_1(A_3)\,.$$

where B_1 and C_1 are fresh nonterminals with fan-out 2. The production r_1 has rank one, so it does not require any further transformations. The transformation thus continues with q_1. The operation g_1 matches the pattern of Case 2, and induces the operations

$$g_2 = x_{1,1}\, \$\, x_{1,2}\,, \quad h_2 = x_{2,1}\$\,, \quad f'_2 = \mathit{wrap}_1\,.$$

The productions constructed from these are

$$p'_2 = B_1 \to \mathit{wrap}_1(B_2, C_2)\,,$$
$$q_2 = B_2 \to g_2(A_1)\,, \quad r_2 = C_2 \to h_2(A_2)\,,$$

where B_2 and C_2 are fresh nonterminals with fan-out 2. At this point, the transformation terminates. We can now delete p from the original grammar, and replace it with the productions $\{p'_1, r_1, p'_2, q_2, r_2\}$.

4.5 Correctness

To see that the transformation is correct, we need to verify that each production of the original grammar is transformed into a set of equivalent normal-form productions, and that the fan-out of the new grammar does not exceed the fan-out of the old grammar.

For the first point, we note that the transformation preserves well-nestedness, decreases the rank of a production, and is always applicable as long as the

rank of a production is at most 2 and the production does not use a concatenation or wrapping operation. That the new productions are equivalent to the old ones in the sense of Figure 3 can be proved by induction on the length of a derivation in the original and the new grammar, respectively.

Let us now convince ourselves that the fan-out of the new grammar does not exceed the fan-out of the old grammar. This is clear in Case 1, where

$$\varphi(f) \;=\; \varphi(g) + \varphi(h) - 1$$

implies that both $\varphi(g) \leq \varphi(f)$ and $\varphi(h) \leq \varphi(f)$. For Case 2, we reason as follows. The fan-out of the operation h, being constructed from an infix of the characteristic string of the original operation f, is clearly bounded by the fan-out of f. For g, we have

$$\varphi(g) \;=\; \varphi(f) - \varphi(h) + 2\,,$$

Now suppose that the index j was chosen according to the first alternative. In this case, $\varphi(h) \geq 2$, and

$$\varphi(g) \;\leq\; \varphi(f) - 2 + 2 \;=\; \varphi(f)\,.$$

For the case where j was chosen according to the second alternative, $\varphi(f) < k$ (since there are no immediate repetitions of gap markers), $\varphi(h) = 1$, and $\varphi(g) \leq k$. If we assume that each nonterminal is productive, then this means that the underlying LCFRS has at least one production with fan-out k or more; therefore, the fan-out of g does not increase the fan-out of the original grammar.

4.6 Complexity

To conclude, we now briefly discuss the space complexity of the normal-form transformation. We measure it in terms of the *length* of a production, defined as the length of its string representation, that is, the string $A \to [v_1 \,\$\, \cdots \,\$\, v_k](A_1, \ldots, A_m)$.

Looking at Figure 3, we note that the normal-form transformation of a production p can be understood as the construction of a (not necessarily complete) binary-branching tree whose leaves correspond to the productions obtained by splitting the characteristic string of p and whose non-leaf nodes are labeled with concatenation and wrapping operations. By construction, the sum of the lengths of leaf-node productions is $\mathcal{O}(|p|)$. Since the number of inner nodes of a binary tree with n leaves is bounded by $n - 1$, we know that the tree has $\mathcal{O}(\rho(p))$ inner nodes. As these nodes correspond to concatenation and wrapping operations, each inner-node production has length $\mathcal{O}(\varphi(p))$. Thus, the sum of the lengths of the productions created from $|p|$ is $\mathcal{O}(|p| + \rho(p)\varphi(p))$. Since the rank of a production is always smaller than its length, this is reduced to $\mathcal{O}(|p|\varphi(p))$.

Therefore, the size of the normal-form transformation of an LCFRS G of fan-out φ is $\mathcal{O}(\varphi|G|)$ in the worst case, and linear space in practice, since the fan-out is typically bounded by a small integer. Taking the normal-form transformation into account, our parser therefore runs in time $\mathcal{O}(\varphi \cdot |G| \cdot |w|^{2\varphi+2})$ where $|G|$ is the original grammar size.

5 Conclusion

In this paper, we have presented an efficient parsing algorithm for well-nested linear context-free rewriting systems, based on a new normal form for this formalism. The normal form takes up linear space with respect to grammar size, and the algorithm is based on a bottom-up process that can be applied to any LCFRS, achieving $\mathcal{O}(\varphi \cdot |G| \cdot |w|^{2\varphi+2})$ time complexity when applied to LCFRS of fan-out φ in our normal form. This complexity is an asymptotic improvement over existing results for this class, both from parsers specifically geared to well-nested LCFRS or equivalent formalisms (Hotz and Pitsch, 1996) and from applying general LCFRS parsing techniques to the well-nested case (Seki et al., 1991).

The class of well-nested LCFRS is an interesting syntactic formalism for languages with discontinuous constituents, providing a good balance between coverage of linguistic phenomena in natural language treebanks (Kuhlmann and Nivre, 2006; Maier and Lichte, 2009) and desirable formal properties (Kanazawa, 2009). Our results offer a further argument in support of well-nested LCFRS: while the complexity of parsing general LCFRS depends on two dimensions (rank and fan-out), this bidimensional hierarchy collapses into a single dimension in the well-nested case, where complexity is only conditioned by the fan-out.

Acknowledgments Gómez-Rodríguez has been supported by MEC/FEDER (HUM2007-66607-C04) and Xunta de Galicia (PGIDIT07SIN005206PR, Redes Galegas de PL e RI e de Ling. de Corpus, Bolsas Estadías INCITE/FSE cofinanced). Kuhlmann has been supported by the Swedish Research Council.

References

Éric Villemonte de la Clergerie. 2002. Parsing mildly context-sensitive languages with thread automata. In *19th International Conference on Computational Linguistics (COLING)*, pages 1–7, Taipei, Taiwan.

Daniel Gildea. 2010. Optimal parsing strategies for linear context-free rewriting systems. In *Human Language Technologies: The Eleventh Annual Conference of the North American Chapter of the Association for Computational Linguistics*, Los Angeles, USA.

Carlos Gómez-Rodríguez, Marco Kuhlmann, Giorgio Satta, and David J. Weir. 2009. Optimal reduction of rule length in linear context-free rewriting systems. In *Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 539–547, Boulder, CO, USA.

Jiří Havelka. 2007. Beyond projectivity: Multilingual evaluation of constraints and measures on non-projective structures. In *45th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 608–615.

Günter Hotz and Gisela Pitsch. 1996. On parsing coupled-context-free languages. *Theoretical Computer Science*, 161(1–2):205–233.

Riny Huybregts. 1984. The weak inadequacy of context-free phrase structure grammars. In Ger de Haan, Mieke Trommelen, and Wim Zonneveld, editors, *Van periferie naar kern*, pages 81–99. Foris, Dordrecht, The Netherlands.

Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. 1975. Tree Adjunct Grammars. *Journal of Computer and System Sciences*, 10(2):136–163.

Aravind K. 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.

Laura Kallmeyer and Wolfgang Maier. 2009. An incremental Earley parser for simple range concatenation grammar. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT 2009)*, pages 61–64. Association for Computational Linguistics.

Makoto Kanazawa and Sylvain Salvati. 2010. The copying power of well-nested multiple context-free grammars. In *Fourth International Conference on Language and Automata Theory and Applications*, Trier, Germany.

Makoto Kanazawa. 2009. The pumping lemma for well-nested multiple context-free languages. In *Developments in Language Theory. 13th International Conference, DLT 2009, Stuttgart, Germany, June 30–July 3, 2009. Proceedings*, volume 5583 of *Lecture Notes in Computer Science*, pages 312–325.

Marco Kuhlmann and Mathias Möhl. 2007. Mildly context-sensitive dependency languages. In *45th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 160–167.

Marco Kuhlmann and Joakim Nivre. 2006. Mildly non-projective dependency structures. In *21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL), Main Conference Poster Sessions*, pages 507–514, Sydney, Australia.

Marco Kuhlmann and Giorgio Satta. 2009. Treebank grammar techniques for non-projective dependency parsing. In *Twelfth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 478–486, Athens, Greece.

Wolfgang Maier and Timm Lichte. 2009. Characterizing discontinuity in constituent treebanks. In *14th Conference on Formal Grammar*, Bordeaux, France.

Wolfgang Maier and Anders Søgaard. 2008. Treebanks and mild context-sensitivity. In *13th Conference on Formal Grammar*, pages 61–76, Hamburg, Germany.

Owen Rambow and Giorgio Satta. 1999. Independent parallelism in finite copying parallel rewriting systems. *Theoretical Computer Science*, 223(1–2):87–120.

Giorgio Satta. 1992. Recognition of Linear Context-Free Rewriting Systems. In *30th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 89–95, Newark, DE, USA.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On Multiple Context-Free Grammars. *Theoretical Computer Science*, 88(2):191–229.

Stuart M. Shieber, Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36.

Stuart M. Shieber. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–343.

Klaas Sikkel. 1997. *Parsing Schemata: A Framework for Specification and Analysis of Parsing Algorithms*. Springer.

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 104–111, Stanford, CA, USA.