LCFRS binarization and debinarization for directional parsing

Wolfgang Maier
Universität Düsseldorf
Institut für Sprache und Information
Universitätsstr. 1, 40225 Düsseldorf, Germany
maierw@hhu.de

Abstract

In data-driven parsing with Linear Context-Free Rewriting System (LCFRS), markovized grammars are obtained through the annotation of binarization non-terminals during grammar binarization, as in the corresponding work on PCFG parsing. Since there is indication that directional parsing with a non-binary LCFRS can be faster than parsing with a binary LCFRS, we present a debinarization procedure with which we can obtain a non-binary LCFRS from a previously binarized one. The resulting grammar retains the markovization information. The algorithm has been implemented and successfully applied to the German NeGra treebank.

1 Introduction

Linear Context-Free Rewriting System (LCFRS), an extension of CFG in which non-terminals can cover more than a single span, can be used to model discontinuous constituents as well as non-projective dependencies (Maier and Lichte, 2011). It has therefore recently been exploited for direct data-driven parsing of such structures. See, e.g., Maier and Søgaard (2008), Maier and Kallmeyer (2010), van Cranenburgh (2011), van Cranenburgh et al. (2012), and Kallmeyer and Maier (2013).

A major problem with data-driven probabilistic LCFRS (PLCFRS) parsing is the high parsing complexity. Given a binary grammar, CYK parsing can be done in $\mathcal{O}(n^{3k})$ where k is the *fan-out* of the grammar, that is, the maximum number of spans that a single non-terminal can cover (Seki et al., 1991). While for PCFG parsing, k is 1, for PLCFRS, k will typically be ≈ 5. Sentences with lengths around 23 to 25 words require very high, unpractical parsing times (20 minutes and more per sentence) (van Cranenburgh et al., 2011; Kallmeyer and Maier, 2013).

One possibility to obtain faster parsing is to reduce the fan-out of the grammar by reducing the number of gaps in the trees from which the grammar is extracted. This has been done by Maier et al. (2012) who transform the trees of the German TIGER treebank such that a grammar fan-out of 2 is guaranteed. For unrestricted PLCFRS parsing, other solutions have been implemented. The parser of Kallmeyer and Maier (2013)[1] offers A^* parsing with outside estimates (Klein and Manning, 2003a). With this technique the practical sentence length limit is shifted upwards by 7 to 10 words. Kallmeyer and Maier also propose non-admissible estimates which provide a greater speed-up but also let the results degrade. The parser of van Cranenburgh (2012)[2] also does not maintain exact search. It implements a coarse-to-fine strategy in a more general PCFG is created from the treebank PLCFRS using the algorithm of Barthélemy et al. (2001). The PCFG chart is then used to filter the PLCFRS chart. While this approach in principle removes the limit[3] on sentence length, it also leads to degraded results.

Yet another solution for obtaining higher speeds could be to turn to parsing strategies which allow for the use of non-binary rules, such as directional CYK parsing or Earley parsing. The reasoning behind this assumption is as follows. Firstly, the longer the right-hand side of a rule, the easier it is to check during parsing on a symbolic basis if it is possible to use the rule

[1] See http://phil.hhu.de/rparse.
[2] See http://github.com/andreasvc/disco-dop.
[3] For very long sentences, i.e. $>$ 60 words, the parser still has extreme memory requirements.

in a complete parse or not by checking the requirements of the rule with respect to the yet unparsed part of the input such as it is done, e.g., in Kallmeyer and Maier (2009).[4] A comparable strategy, the F grammar projection estimate (Klein and Manning, 2003a), has been employed in PCFG parsing. Secondly, practical experience with Range Concatenation Grammar (RCG) (Boullier, 1998) and Grammatical Framework (GF) (Ranta, 2004) parsing points in the same direction. Lazy computation of instantiations in RCG parsing as done in the TuLiPA system (Kallmeyer et al., 2009) and the SYNTAXE parser (Boullier and Deschamp, 1988) (Boullier, p.c.) seem to be less effective with shorter right-hand sides of rules because less constraints can be collected at once. Practical experiments with the GF[5] parser (Angelov, 2009), which implements an Earley strategy, indicate that certain optimizations loose their effect with binary grammars (Angelov, p.c.).

So why not just do directional parsing with the unbinarized treebank grammar? It is common knowledge that vanilla treebank PCFGs do not perform well. This also holds for PLCFRS. Markovization has been proven to be an effective remedy for both PCFG and PLCFRS parsing. It can be achieved through the probability model itself (Collins, 1999) or by annotating the treebank grammar (Klein and Manning, 2003b; Kallmeyer and Maier, 2013): Instead of using a unique non-terminal as in deterministic binarization, one uses a single non-terminal and adorns it with the vertical ("parent annotation", see Johnson (1998)) and horizontal context of the occurrence of the rule in the treebank. This augments the coverage of the grammar and helps to achieve better parsing results by adding a possibly infinite number of implicit non-binary rules.

Our main contribution in this article is a *debinarization* algorithm with which a non-binary LCFRS can be generated from a previously binarized LCFRS (which fulfills certain conditions). Given a markovized binarized grammar, the debinarized grammar contains non-binary productions obtained through markovization. We furthermore contribute a new compact notation for rules of a *treebank LCFRS*, i.e., of the variant of LCFRS obtained by treebank grammar extraction, and provide a formulation of deterministic binarization using this notation.

An implementation of the debinarization algorithm has been tested on the German NeGra treebank (Skut et al., 1997). First experimental results confirm that in practice, the debinarized grammar can perform better than than the plain treebank grammar.

The remainder of the article is structured as follows. In the following section, we define treebank LCFRS and introduce our new rule representation. We furthermore introduce the binarization, resp. markovization algorithm. In section 3, we introduce our new debinarization algorithm. Section 4 presents the experimental evaluation and section 5 closes the article.

2 LCFRS

2.1 Definition

In LCFRS (Vijay-Shanker et al., 1987), a single non-terminal can span $k \geq 1$ continuous blocks of a string. A CFG is simply a special case of an LCFRS in which $k = 1$. k is called the *fan-out* of the non-terminal. We notate LCFRS with a syntax of Simple Range Concatenation Grammars (SRCG) (Boullier, 1998), a formalism equivalent to LCFRS. In the following we define *treebank LCFRS*, the variant of LCFRS which is obtained by the grammar extraction algorithm of Maier and Søgaard (2008).

A *treebank LCFRS* is a tuple $G = (N, T, V, P, S)$ where

1. N is a finite set of non-terminals with a function $dim\colon N \to \mathbb{N}$ determining the *fan-out* of each $A \in N$;
2. T and V are disjoint finite sets of terminals and variables;
3. $S \in N$ is the start symbol with $dim(S) = 1$;
4. P is a finite set of rewriting rules where all $r \in P$ are either
   (a) rules with rank $m \geq 1$ of the form
   $$A(\alpha_1, \ldots, \alpha_{dim(A)}) \to A_1(X_1^{(1)}, \ldots, X_{dim(A_1)}^{(1)}) \cdots A_m(X_1^{(m)}, \ldots, X_{dim(A_m)}^{(m)})$$
   where
   i. $A, A_1, \ldots, A_m \in N$, $X_j^{(i)} \in V$ for $1 \leq i \leq m, 1 \leq j \leq dim(A_i)$ and

[4] For such a check, it makes no difference if the rule is deterministically binarized.

[5] http://grammaticalframework.org

$A(a) \to \varepsilon$	($\langle a \rangle$ in yield of A)
$B(b) \to \varepsilon$	($\langle b \rangle$ in yield of B)
$C(X,Y) \to A(X)A(Y)$	(if $\langle X \rangle$ in the yield of A and $\langle Y \rangle$ in the yield of A, then $\langle X, Y \rangle$ in yield of C)
$S(XZY) \to C(X,Y)B(Z)$	(if $\langle X, Y \rangle$ in yield of C and $\langle Z \rangle$ in the yield of B, then $\langle XZY \rangle$ in yield of S)

$L = \{a^n b a^n \mid n > 0\}$

Figure 1: Yield example

ii. $\alpha_i \in V^+$ for $1 \leq i \leq dim(A)$ (we write $\alpha_i.j$, $1 \leq j \leq |\alpha_i|$ for the jth variable in α_i), or

(b) rules of rank 0 of the form $A(t) \to \varepsilon$ where $A \in N, t \in T$.

For all $r \in P$, every variable X that occurs in r occurs exactly once in the left-hand side (LHS) and exactly once in the right-hand side (RHS). Furthermore, if for two variables $X_1, X_2 \in V$, it holds that $X_1 \prec X_2$ on the RHS, then also $X_1 \prec X_2$ on the LHS.[6]

A rewriting rule describes how to compute the yield of the LHS non-terminal from the yields of the RHS non-terminals. The yield of S is the language of the grammar. See figure 1 for an example.

The *rank* of G is the maximal rank of any of its rules, its *fan-out* is the maximal fan-out of any of its non-terminals.

The properties which distinguish a treebank LCFRS from a regular LCFRS are the requirements that

1. all non-terminal arguments are variables except in terminating lexical rules in which the only argument of the LHS consists of a single terminal, and

2. the ordering property.

These properties allows us to notate rules in a more compact way. Let (N, T, V, P, S) be an LCFRS. A rule $r \in P$

$$A(\alpha_1, \ldots, \alpha_{dim(A)}) \to A_1(X_1^{(1)}, \ldots, X_{dim(A_1)}^{(1)}) \cdots A_m(X_1^{(m)}, \ldots, X_{dim(A_m)}^{(m)})$$

can be represented as a tuple $(A \to A_1 \cdots A_m, \vec{\varphi})$, where $\vec{\varphi}$ is a *linearization vector* which is defined as follows. For all $1 \leq i \leq dim(A)$, $1 \leq j \leq |\alpha_i|$, $\vec{\varphi}[i]$ is a *subvector* of $\vec{\varphi}$ with $|\vec{\varphi}[i]| = |\alpha_i|$ and it holds that $\vec{\varphi}[i][j] = x$ iff $\alpha_i.j$ occurs in the arguments of A_x. Let us consider an example: The rule $A(X_1X_2, X_3X_4) \to B(X_1, X_3)C(X_2, X_4)$ can be fully specified by $(A \to BC, [[1,2],[1,2]])$.

We introduce additional notation for operations on a linearization vector $\vec{\varphi}$.

1. We write $\vec{\varphi}/(x, x')$ for some $x, x' \in \mathbb{N}$ for the substitution of all occurrences of x in $\vec{\varphi}$ by x'. $\vec{\varphi}/^s(x, x')$ denotes the same substitution procedure with the addition that after the substitution, all occurrences of x'^+ in $\vec{\varphi}$ are replaced by x'.

2. We write $\vec{\varphi} \setminus x$ for some $x \in \mathbb{N}$ for the deletion of all occurrences of x in $\vec{\varphi}$, followed by the deletion of subvectors which become empty.

3. $\vec{\varphi} \downarrow^x$ for some $x \in \mathbb{N}$ denotes the splitting of all subvectors of $\vec{\varphi}$ such that a vector boundary is introduced between all $\vec{\varphi}[i][j]$ and $\vec{\varphi}[i][j+1]$, $1 \leq i \leq |\vec{\varphi}|$, $1 \leq j \leq |\vec{\varphi}[i]| - 1$ iff $\vec{\varphi}[i][j] = x$.

Note that for every LCFRS, there is an equivalent LCFRS which fulfills the treebank LCFRS conditions (Kallmeyer, 2010).

A *probabilistic (treebank) LCFRS* (PLCFRS) is a tuple $\langle N, T, V, P, S, p \rangle$ such that $\langle N, T, V, P, S \rangle$ is a (treebank) LCFRS and $p : P \to [0..1]$ a function such that for all $A \in N$: $\Sigma_{A(\vec{x}) \to \vec{\Phi} \in P} p(A(\vec{x}) \to \vec{\Phi}) = 1$. The necessary counts for a Maximum Likelihood estimation can easily be obtained from the treebank (Kallmeyer and Maier, 2013).

2.2 Binarization and Markovization

Using the linearization vector notation, deterministic left-to-right binarization of treebank LCFRS can be accomplished as in algorithm 1. The algorithm creates binary top and bottom rules but can easily be extended to create unary rules in these places. Note that the algorithm behaves identically to other deterministic binarization algorithms such as, e.g., the one in Kallmeyer (2010).

The algorithm works as follows. We start with the linearization vector $\vec{\varphi}$ of the original rule. In each binarization step, we "check off" from $\vec{\varphi}$ the information which was used in the previous step. The

[6] This is the *ordering* property which Villemonte de la Clergerie (2002) defines for RCGs. LCFRSs with this property are called *monotone* (Michaelis, 2001), MCFGs *non-permuting* (Kracht, 2003).

Algorithm 1 LCFRS binarization

Let (N, T, V, P, S) be a treebank LCFRS
$P' = \emptyset$
for all $(A \to A_1 \dots A_m, \vec{\varphi}_r)$ in P with $m > 2$ **do**
  pick new non-terminals $C_1, \dots, C_{m-2}$
  let $\vec{\varphi} = \vec{\varphi}_r$
  let $\vec{\varphi}_g = \vec{\varphi}_r / ^s(x, 2)$ for all $x > 1$
  add the rule $A \to A_1 C_1, \vec{\varphi}_g$ to P'
  for all i, $1 \leq i < m - 2$ **do**
    let $\vec{\varphi} = \vec{\varphi}/(x, x-1)$ for all x
    let $\vec{\varphi} = (\vec{\varphi} \downarrow^0) \setminus 0$
    let $\vec{\varphi}_g = \vec{\varphi} / ^s(x, 2)$ for all $x > 1$
    add the rule $(C_i \to A_{i+1} C_{i+1}, \vec{\varphi}_g)$ to P'
  let $\vec{\varphi} = \vec{\varphi}/(x, x-1)$ for all x
  let $\vec{\varphi} = (\vec{\varphi} \downarrow^0) \setminus 0$
  add the rule $(C_{m-2} \to A_{m-1} A_m, \vec{\varphi})$ to P'
set P to P'

linearization vectors of the binary rules are obtained from the respective current state of $\vec{\varphi}$ by attributing all information which will covered by the new binarization non-terminal to exactly this non-terminal.

As an example, consider the binarization of the rule $A(X_1X_2, X_3X_4, X_5X_6) \to B(X_1, X_3)C(X_2, X_4)D(X_5)E(X_6)$. The compact representation of the rule is $(A \to BCDE, [[1,2][1,2][3,4]])$. The first step yields the rule $(A \to BC_1, [[1,2],[1,2],[2]])$. $\vec{\varphi}$ is set to the linearization vector of the original rule. Subsequently, we remove from $\vec{\varphi}$ the information which has already been used by subtracting one from all integers in $\vec{\varphi}$, splitting the vector at occurrences of 0 (i.e., new argument boundaries are introduced at positions which have been covered by the previous binarization step), and then removing all occurrences of 0. From the resulting vector $[[1],[1],[2,3]]$, we create the linearization vector for the second binary rule $(C_1 \to CC_2, [[1],[1],[2]])$ by attributing all material covered by C_2 to C_2. In other words, the last subvector can be reduced from $[2,3]$ to $[2]$ since only in the next and last binarization step we will distinguish between 2 and 3. In the subsequent last step, we again check off the already used material from $\vec{\varphi}$ and end up with $(C_2 \to DE, [[1,2]])$.

During binarization, a grammar can be markovized by changing the choice of binarization non-terminals. Instead of unique non-terminals $C_1, \dots, C_{m-2}$, we pick a single non-terminal @. At each binarization step, to this non-terminal, we append the vertical occurrence context from the treebank, i.e., we perform parent annotation (Johnson, 1998), and we append the horizontal context. More precisely, the markovization information for the binarization non-terminal that comprises original RHS elements $A_i \dots A_m$ are the first v elements of path from A_i to root vertically and the first h elements of $A_i \dots A_0$ horizontally.

3 Debinarization

The goal of the debinarization procedure is to obtain a grammar with non-binary rules from a previously binarized one. Algorithm 2 accomplishes the debinarization. It consists of a function *debin*, which, assuming a treebank LCFRS (N, T, V, P, S) binarized with algorithm 1, is called on all $A \in N$. During debinarization, linearization vectors must be recombined. This is notated as function *lin*.

Called on some non-terminal $A \in N$, the algorithm recursively substitutes all binarization non-terminals on the right-hand sides of all binary A rules with the right-hand sides of rules which have the same binarization non-terminal on its LHS. The base case of this recursion are rules which do not have binarization non-terminals on their right-hand side. At each substitution, we recombine the corresponding linearization vectors: When substituting the binarization non-terminal C on the right-hand side of a binary production by the RHS of a debinarized rule r with C on its LHS, roughly, we replace the ith occurrence of 2 in the linearization vector by the ith subvector of the linearization vector of r, adding 1 to all of its elements.[7]

The probability of a rule in which a binarization non-terminal on the RHS is substituted by the RHS of a rule with this binarization non-terminal on its LHS is simply the product of both probabilities.

As an example, consider the debinarization of the non-terminal A given the binarized rules from our previous example. We have a single A rule, in which we would recursively substitute the C_1 on the right-hand side with $C\,C_2$ and in the following as base case C_2 with $D\,E$. In the base case, **lin** is not called, be-

[7] The algorithm would be slightly more complex if, during binarization, we would permit that binarization non-terminals end up on the left corner of the right-hand side of a binary production. Algorithm 1 guarantees that binarization non-terminals occur only on the right corner.

cause no linearization vector recombination is necessary. For the substitution of C_2, we combine the linearization vectors of the C_1 and C_2 rules. For this, we first add 1 to all elements of the C_2 vector, which gets us $[[2,3]]$. We then replace the only occurrence of 2 in the C_1 vector by $[2,3]$. The result is the vector $[[1],[1],[2,3]]$. For the substitution of C_1 in the A rule, we recombine our result vector with the vector of the A rule. For this, we first add 1 to all elements of the result of the vector recombination, which gets us $[[2],[2],[3,4]]$. We then replace all three occurrences of 2 in the vector $[[1,2],[1,2],[2]]$ of the binary A rule with the corresponding subvectors of $[[2],[2],[3,4]]$. This give us $[[1,2],[1,2],[3,4]]$, which is the linearization vector of the original unbinarized A rule.

In the case of deterministic binarization, the algorithm yields a grammar which is equivalent to the grammar before the binarization. With markovization, i.e., with non-deterministic binarization, it is more difficult. Since we do not choose our binarization nonterminals in a unique way, it is possible that a chain of substitutions can lead us in a cycle, i.e., that we reach the same binarization non-terminal again. A cycle corresponds to an infinite number of implicit nonbinary rules. The smaller the binarization context, the more likely it is that such a cycle occurs.

Several strategies are possible to avoid an infinite loop of substitutions. The most obvious one is to use a cache which contains all binarization nonterminals seen during debinarization. If a substitution introduces a binarization non-terminal which is in the cache, the corresponding production is discarded. Another strategy is to discard rules with a probability lower than a certain threshold.

4 Experiments

We have implemented the grammar extraction algorithm from Maier and Søgaard (2008), as well as the binarization and the debinarization algorithm. We implement the debinarization algorithm with a probability filter: A substitution of a binarization non-terminal will not take place if the probability of the resulting production falls under a certain threshold. We furthermore implement a filter which excludes rules with right-hand sides that exceed a certain length.

In order to experimentally test the debinarization algorithm, we perform experiments on the NeGra treebank (Skut et al., 1997) using rparse.[8] In a first step, we apply the algorithm for re-attaching elements to the virtual root node described in (Maier et al., 2012). All sentences longer than 20 words are excluded. For parsing, we split the data and use the first 90% of all sentences for training and the remainder for parsing. We then extract the grammar from the training part (results in 10,482 rules) and binarize them, using deterministic binarization and using markovization with vertical and horizontal histories of 1, resp. 2. The markovized grammar is debinarized, with an experimentally determined logarithmic rule weight of 15 for the filtering and a limit of 15 on the lengths of rule right-hand sides. This results in a grammar with 175,751 rules.

We then parse with the deterministically binarized treebank grammar, with the markovized binary grammar, and with debinarized grammar. Note that rparse currently offers no directional parser. Therefore we rebinarize the debinarized grammar using deterministic binarization. Using bracket scoring (evalb)[9], we obtain the 75.49 F_1 for the plain treebank grammar,

Algorithm 2 LCFRS debinarization

function $\mathbf{debin}(A \in N)$
let $\mathcal{R} = \emptyset$
for all $(A \to A_1 A_2, \vec{\varphi})$ in P **do**
  if A_1, A_2 are not bin. non-terminals **then**
    add $(A \to A_1 A_2, \vec{\varphi})$ to $\mathcal{R}$
  else
    let $\mathcal{D} = \mathbf{debin}(A_2)$
    for all $(D \to D_0 \cdots D_m, \vec{\varphi}_D) \in \mathcal{D}$ **do**
      add $(A \to A_1 D_0 \cdots D_m, \mathbf{lin}(\vec{\varphi}, \vec{\varphi}_D, 2))$ to $\mathcal{R}$
return $\mathcal{R}$

function $\mathbf{lin}(\vec{\varphi}, \vec{\varphi}_D, s)$
let $c = 0$
let $\vec{\varphi}_D = \vec{\varphi}_D / (x, x+1)$ for all x
for all occurrences of s in $\vec{\varphi}$ **do**
  replace occ. of s with the content of $\vec{\varphi}_D[c]$
  $c = c + 1$
return $\vec{\varphi}$

[8] See http://phil.hhu.de/rparse.
[9] See http://github.com/wmaier/evalb-lcfrs.

77.10 for the markovized grammar and 76.37 for the debinarized grammar. This shows that the debinarized markovized grammar can perform better than the plain treebank grammar. However, the large number of productions in the debinarized grammar indicates that the possibilities of filtering must be investigated more closely. This is left for future work.

5 Conclusion

We have presented a new compact representation for grammar rules of a treebank LCFRS together with a formulation of a binarization algorithm. Furthermore, we have presented a procedure for debinarizing a previously binarized LCFRS. The resulting grammar maintains the markovization information introduced during binarization and can be used with directional parsing strategies. Experiments have shown that the grammar can perform better than the plain treebank grammar. There remains potential for optimization.

We are currently working on the integration of the algorithm into a directional parsing strategy within a data-driven PLCFRS parser.

Acknowledgments

I would like Laura Kallmeyer, Miriam Kaeshammer and the three reviewers for comments and suggestions.

References

Krasimir Angelov. 2009. Incremental parsing with Parallel Multiple Context-Free Grammars. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 69–76, Athens, Greece.

François Barthélemy, Pierre Boullier, Philippe Deschamp, and Éric Villemonte de la Clergerie. 2001. Guided parsing of Range Concatenation Languages. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 42–49, Toulouse, France.

Pierre Boullier and Philippe Deschamp, 1988. *Le système SYNTAX^{TM} – manuel d'utilisation et de mise en oeuvre sous UNIX^{TM}*. http://syntax.gforge.inria.fr/syntax3.8-manual.pdf, January 4, 2012.

Pierre Boullier. 1998. A generalization of mildly context-sensitive formalisms. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+4)*, pages 17–20, Philadelphia, PA.

Michael Collins. 1999. *Head-driven statistical models for natural language parsing*. Ph.D. thesis, University of Pennsylvania.

Mark Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

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'09)*, pages 61–64, Paris, France. Association for Computational Linguistics.

Laura Kallmeyer and Wolfgang Maier. 2013. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*, 39(1):87–119.

Laura Kallmeyer, Wolfgang Maier, and Yannick Parmentier. 2009. An Earley parsing algorithm for Range Concatenation Grammars. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 9–12, Singapore.

Laura Kallmeyer. 2010. *Parsing Beyond Context-Free Grammars*. Springer.

Dan Klein and Christopher D. Manning. 2003a. A* Parsing: Fast exact viterbi parse selection. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 40–47, Edmonton, Canada.

Dan Klein and Christopher D. Manning. 2003b. Accurate unlexicalized parsing. In *Proceedings of the 41th Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan. Association for Computational Linguistics.

Marcus Kracht. 2003. *The Mathematics of Language*. Mouton de Gruyter, Berlin.

Wolfgang Maier and Laura Kallmeyer. 2010. Discontinuity and non-projectivity: Using mildly context-sensitive formalisms for data-driven parsing. In *Proceedings of the Tenth International Conference on Tree Adjoining Grammars and Related Formalisms (TAG+10)*, pages 119–126, Yale University, New Haven, CT.

Wolfgang Maier and Timm Lichte. 2011. Characterizing discontinuity in constituent treebanks. In *Formal Grammar. 14th International Conference, FG 2009. Bordeaux, France, July 25-26, 2009. Revised Selected*

Papers, volume 5591 of *Lecture Notes in Artificial Intelligence*, pages 167–182, Berlin/Heidelberg/New York. Springer-Verlag.

Wolfgang Maier and Anders Søgaard. 2008. Treebanks and mild context-sensitivity. In Philippe de Groote, editor, *Proceedings of the 13th Conference on Formal Grammar (FG-2008)*, pages 61–76, Hamburg, Germany. CSLI Publications.

Wolfgang Maier, Miriam Kaeshammer, and Laura Kallmeyer. 2012. Data-driven PLCFRS parsing revisited: Restricting the fan-out to two. In *Proceedings of the Eleventh International Conference on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, Paris, France. to appear.

Jens Michaelis. 2001. *On Formal Properties of Minimalist Grammars*. Ph.D. thesis, Potsdam University, Potsdam, Germany.

Aarne Ranta. 2004. Grammatical Framework, a typetheoretical grammar formalism. *Journal of Functional Programming*, 14(2):145–189.

Hiroyuki Seki, Takahashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. An annotation scheme for free word order languages. In *Proceedings of the 5th Applied Natural Language Processing Conference*, pages 88–95, Washington, DC.

Andreas van Cranenburgh, Remko Scha, and Federico Sangati. 2011. Discontinuous data-oriented parsing: A mildly context-sensitive all-fragments grammar. In *Proceedings of the Second Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2011)*, pages 34–44, Dublin, Ireland.

Andreas van Cranenburgh. 2012. Efficient parsing with linear context-free rewriting systems. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 460–470, Avignon, France.

K. Vijay-Shanker, David Weir, and Aravind K. Joshi. 1987. Characterising structural descriptions used by various formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Stanford, CA. Association for Computational Linguistics.

Éric Villemonte de la Clergerie. 2002. Parsing Mildly Context-Sensitive Languages with Thread Automata. In *Proceedings of COLING 2002: The 19th International Conference on Computational Linguistics*, Taipei, Taiwan.