Data-driven Parsing using PLCFRS

Data-driven Parsing using Probabilistic Linear Context-Free Rewriting Systems

Laura Kallmeyer[*]
University of Düsseldorf

Wolfgang Maier[**]
University of Düsseldorf

This paper presents the first efficient implementation of a weighted deductive CYK parser for Probabilistic Linear Context-Free Rewriting Systems (PLCFRS). LCFRS, an extension of CFG, can describe discontinuities in a straightforward way and is therefore a natural candidate to be used for data-driven parsing. To speed up parsing, we use different context-summary estimates of parse items, some of them allowing for A^* parsing. We evaluate our parser with grammars extracted from the German NeGra treebank. Our experiments show that data-driven LCFRS parsing is feasible and yields output of competitive quality.

1. Introduction

1.1 Discontinuous Constituents

In the context of data-driven parsing, challenges always arise from discontinous constituents since they call for formalisms which have a larger domain of locality than context-free grammars. Discontinuities are particularly frequent in so-called free word order languages such as German. A few examples are shown in (1). (1a) shows several examples for discontinuous VPs while (1b) contains a discontinuous NP.

(1) a. Fronting:
Darüber muss *nachgedacht* werden. (NeGra)
Thereof must *thought* be
"One must think of that"

Ohne internationalen Schaden könne *sich* Bonn *von dem Denkmal* nicht *distanzieren*, ... (TIGER)
Without international damage could *itself* Bonn *from the monument* not *distance*
"Bonn could not distance itself from the monument without international damage."

Auch würden *durch die Regelung* nur *"ständig* neue Altfälle *entstehen"*. (TIGER)
Also would *through the regulation* only *"constantly* new old cases *emerge"*
"Apart from that, the regulation would only constantly produce new old cases."

b. Extraposed relative clauses:

[*] SFB 991, Universitätsstr. 1, D-40225 Düsseldorf, Germany. E-mail: kallmeyer@phil.uni-duesseldorf.de

[**] SFB 991, Universitätsstr. 1, D-40225 Düsseldorf, Germany. E-mail: maierw@hhu.de

. . . ob auf deren Gelände der *Typ von Abstellanlage* gebaut werden könne, *der ...* (NeGra)
. . . whether on their terrain the *type of parking facility* built get could, *which . . .*
". . . whether one could build on their premises the type of parking facility, which . . . "

In the German NeGra treebank (Skut et al. 1997), approximately 25% of the sentences display discontinuous constituents. This shows that they are no minor phenomenon one could neglect but that it is important to find a way to adequately deal with them.

Discontinuous constituents are by no means limited to languages such as German. They also occur in languages with a rather fixed word order such as English, resulting for instance from long-distance movements. (2) shows a few examples taken from the Penn Treebank (PTB) (Marcus et al. 1994). (2a.) is a long extraction resulting in a discontinuous S category while (2b.) displays a discontinuous NP arising from an extraposed relative clause.

(2) a. Long Extractions:
Those chains include Bloomingdale's, *which* Campeau recently said *it will sell*.
b. Extraposed relative clauses:
They sow *a row of male-fertile plants* nearby, *which then pollinate the male-sterile plants*.

1.2 PCFG Parsing

Data-driven parsing has largely been dominated by Probabilistic Context-Free Grammar (PCFG). The use of PCFG is tied to the annotation principles of popular treebanks such as the Penn Treebank, which are used as a data source for grammar extraction. Their annotation generally relies on the use of an annotation backbone based on Context-Free Grammar (CFG), augmented with a mechanism that accounts for non-local dependencies. In the PTB, e.g., trace nodes and co-indexation markers are used in order to establish additional implicit edges in the tree beyond the overt phrase structure. In contrast, some other treebanks, such as the German NeGra and TIGER treebanks, give up the annotation backbone based on CFG and allow annotation with crossing branches (Skut et al. 1997). Non-local dependencies can then be expressed directly by grouping all dependent elements under a single node.

However, given the expressivity restrictions of PCFG, work on data-driven parsing has mostly excluded non-local depdendencies. For treebanks with PTB-like annotation, this amounts to discarding the co-indexation and the trace nodes. For treebanks like NeGra and TIGER, tree transformations are applied which resolve the crossing branches (see, e.g., Kübler (2005) and Boyd (2007)). Especially for those treebanks, such a transformation is problematic, since it is non-reversible and implies information loss.

1.3 Extending the Domain of Locality

In the literature, different methods have been explored which allow for the use of non-local information in data-driven parsing. We distinguish two classes of approaches. The first class of approaches aims at using formalisms which produce trees without crossing branches but provide a larger domain of locality than CFG, for instance through complex labels (Hockenmaier 2003) or through the derivation mechanism (Chiang 2003). The second class of approaches aims at producing trees which contain non-local infor-

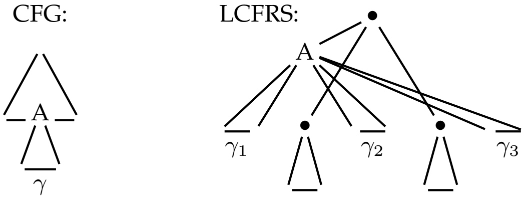


Figure 1
Different domains of locality

mation. Some methods realize the reconstruction of non-local information in a post- or pre-processing step to PCFG parsing (Johnson 2002; Dienes 2003; Levy and Manning 2004). Other work uses formalisms which accommodate the direct encoding of non-local information (Plaehn 2004; Levy 2005). This paper pursues the last approach.

Our work is motivated by the following recent developments. Linear Context-Free Rewriting Systems (LCFRSs) (Vijay-Shanker, Weir, and Joshi 1987) have been established as a candidate for modeling both discontinuous constituents and non-projective dependency trees as they occur in treebanks (Maier and Søgaard 2008; Kuhlmann and Satta 2009; Maier and Lichte 2009). LCFRSs are a natural extension of CFGs where the non-terminals can span tuples of possibly non-adjacent strings (see Fig. 1). Since LCFRSs allow for binarization and CYK chart parsing in a way similar to CFGs, PCFG techniques, such as Best-First Parsing (Charniak and Caraballo 1998), Weighted Deductive Parsing (Nederhof 2003) and A* parsing (Klein and Manning 2003a), can be transferred to LCFRS. Finally, languages such as German have attracted the interest of the parsing community due to the challenges arising from its frequent discontinuous constituents (Kübler and Penn 2008; Seddah, Kübler, and Tsarfaty 2010). LCFRS, precisely because of its extended locality, is a promising candidate for parsing languages such as German.

We bring together these developments by presenting a parser for probabilistic LCFRS, continuing on the promising work of Levy (2005). To our knowledge, our parser is the first for the entire class of PLCFRS which has successfully been used for data-driven parsing. We have implemented a CYK parser and we present several methods for context summary estimation of parse items. The estimates either act as figures-of-merit in a best-first parsing context or as estimates for A* parsing. Last, a test on a real-world-sized data set shows that our parser achieves competetive results.[1]

The paper is structured as follows. Section 2 introduces probabilistic LCFRS. The sections 3 and 4 present the binarization algorithm, the parser and the outside estimates which we use to speed up parsing. In section 5 we explain how to extract an LCFRS from a treebank and we present ways of grammar refinement. Section 6 finally presents evaluation results.

2. Probabilistic Linear Context-Free Rewriting Systems

2.1 Definition of PLCFRS

LCFRS (Vijay-Shanker, Weir, and Joshi 1987) is an extension of CFG in which the non-terminals can span not only single strings but tuples of strings. We will notate LCFRS

[1] Parts of the results presented in this paper have been presented earlier (Kallmeyer and Maier 2010; Maier and Kallmeyer 2010; Maier 2010).

$$\begin{array}{rcll} A(ab, cd) & \rightarrow & \varepsilon & \\ A(aXb, cYd) & \rightarrow & A(X, Y) & \text{Goal: } [S, \langle\langle 0, n\rangle\rangle] \\ S(XY) & \rightarrow & A(X, Y) & \end{array}$$

Figure 2
Sample LCFRS for $\{a^n b^n c^n d^n \mid n \geq 1\}$

with the syntax of *simple Range Concatenation Grammars* (SRCG) (Boullier 1998b), a formalism that is equivalent to LCFRS. A third formalism that is equivalent to LCFRS is *Multiple Context-Free Grammar* (MCFG) (Seki et al. 1991).

Definition 1 (LCFRS)
A *Linear Context-Free Rewriting System* (LCFRS) is a tuple $\langle N, T, V, P, S\rangle$ where

a) N is a finite set of non-terminals with a function dim: $N \rightarrow \mathbb{N}$ that determines the *fan-out* of each $A \in N$;

b) T and V are disjoint finite sets of terminals and variables;

c) $S \in N$ is the start symbol with $dim(S) = 1$;

d) P is a finite set of rules
$A(\alpha_1, \ldots, \alpha_{dim(A)}) \rightarrow A_1(X_1^{(1)}, \ldots, X_{dim(A_1)}^{(1)})$
$\cdots A_m(X_1^{(m)}, \ldots, X_{dim(A_m)}^{(m)})$
for $m \geq 0$ where $A, A_1, \ldots, A_m \in N$, $X_j^{(i)} \in V$ for $1 \leq i \leq m, 1 \leq j \leq dim(A_i)$ and $\alpha_i \in (T \cup V)^*$ for $1 \leq i \leq dim(A)$. For all $r \in P$, it holds that every variable X occurring in r occurs exactly once in the left-hand side and exactly once in the right-hand side of r.

A rewriting rule describes how the yield of the left-hand side non-terminal can be computed from the yields of the right-hand side non-terminals. The rules $A(ab, cd) \rightarrow \varepsilon$ and $A(aXb, cYd) \rightarrow A(X, Y)$ from Fig. 2 for instance specify that 1. $\langle ab, cd\rangle$ is in the yield of A and 2. one can compute a new tuple in the yield of A from an already existing one by wrapping a and b around the first component and c and d around the second. A CFG rule $A \rightarrow BC$ would be written $A(XY) \rightarrow B(X)C(Y)$ as an LCFRS rule.

Definition 2 (Yield, language)
Let $G = \langle N, T, V, P, S\rangle$ be a LCFRS.

1. For every $A \in N$, we define the yield of A, $yield(A)$ as follows:
   a) For every $A(\vec{\alpha}) \rightarrow \varepsilon$, $\vec{\alpha} \in yield(A)$;
   b) For every rule

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

   and all $\vec{\tau_i} \in yield(A_i)$ for $1 \leq i \leq m$:
   $\langle f(\alpha_1), \ldots, f(\alpha_{dim(A)})\rangle \in yield(A)$ where f is defined as follows:
   (i) $f(t) = t$ for all $t \in T$,
   (ii) $f(X_j^{(i)}) = \vec{\tau_i}(j)$ for all $1 \leq i \leq m, 1 \leq j \leq dim(A_i)$ and
   (iii) $f(xy) = f(x)f(y)$ for all $x, y \in (T \cup V)^+$.

We call f the *composition function* of the rule.

c) Nothing else is in $yield(A)$.

2. The language of G is then $L(G) = \{w \mid \langle w \rangle \in yield(S)\}$.

As an example, consider again the LCFRS in Fig. 2. The last rule tells us that, given a pair in the yield of A, we can obtain an element in the yield of S by concatenating the two components. Consequently, the language generated by this grammar is $\{a^n b^n c^n d^n \mid n \geq 1\}$.

The terms fan-out and rank and the properties of being monotone and ε-free will be referred to later and are therefore introduced in the following definition. These terms are taken form the LCFRS/MCFG terminology; the simple RCG term for fan-out is *arity* while the property of being monotone is called *ordered* in the context of simple RCG.

Definition 3
Let $G = \langle N, T, V, P, S \rangle$ be a LCFRS.

1. The *fan-out* of G is the maximal fan-out of all non-terminals in G.
2. Furthermore, the right-hand side length of a rewriting rule $r \in P$ is called the *rank* of r and the maximal rank of all rules in P is called the *rank* of G.
3. G is *monotone* if for every $r \in P$ and every right-hand side non-terminal A in r and each pair X_1, X_2 of arguments of A in the right-hand side of r, X_1 precedes X_2 in the right-hand side iff X_1 precedes X_2 in the left-hand side.
4. A rule $r \in P$ is called an ε-rule if one of the left-hand side components of r is ε.
   G is *ε-free* if it either contains no ε-rules or there is exactly one ε-rule $S(\varepsilon) \to \varepsilon$ and S does not appear in any of the right-hand sides of the rules in the grammar.

For every LCFRS there exists an equivalent LCFRS that is ε-free (Seki et al. 1991; Boullier 1998a) and monotone (Michaelis 2001; Kracht 2003; Kallmeyer 2010).

The definition of a probabilistic LCFRS is a straightforward extension of the definition of PCFG and it follows (Levy 2005; Kato, Seki, and Kasami 2006):

Definition 4 (PLCFRS)
A *probabilistic LCFRS* (PLCFRS) is a tuple $\langle N, T, V, P, S, p \rangle$ such that $\langle N, T, V, P, S \rangle$ is a LCFRS and $p : P \to [0..1]$ a function such that for all $A \in N$:

$$\Sigma_{A(\vec{x})\to\vec{\Phi}\in P}\, p(A(\vec{x}) \to \vec{\Phi}) = 1$$

PLCFRS with non-terminals $\{S, A, B\}$, terminals $\{a\}$ and start symbol S:

$0.2 : S(X) \to A(X)$	$0.8 : S(XY) \to B(X, Y)$
$0.7 : A(aX) \to A(X)$	$0.3 : A(a) \to \varepsilon$
$0.8 : B(aX, aY) \to B(X, Y)$	$0.2 : B(a, a) \to \varepsilon$

Figure 3
Sample PLCFRS

As an example, consider the PLCFRS in Fig. 3. This grammar simply generates a^+. Words with an even number of as and nested dependencies are more probable than words with a right-linear dependency structure. For instance, the word aa receives the two analyses in Fig. 4. The analysis (a) displaying nested dependencies has probability 0.16 while (b) (right-linear dependencies) has probability 0.042.

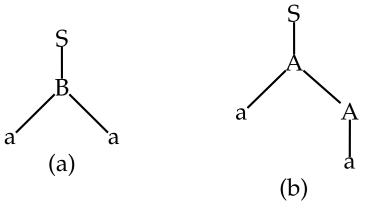


Figure 4
The two derivations of aa

3. Parsing PLCFRS

3.1 Binarization

Similarly to the transformation of a CFG into Chomsky Normal Form (CNF), we binarize the LCFRS extracted from the treebank. The result is an LCFRS of rank 2. As in the CFG case, in the transformation , we introduce a non-terminal for each right-hand side longer than 2 and split the rule into two rules, using this new intermediate non-terminal. This is repeated until all right-hand side are of length 2. The transformation algorithm is inspired by Gómez-Rodríguez et al. (2009) and it is also specified in Kallmeyer (2010).

In order to give the algorithm for this transformation, we need the notion of a *reduction* of a vector $\vec{\alpha} \in [(T \cup V)^*]^i$ by a vector $\vec{x} \in V^j$ where all variables in $\vec{x}$ occur in $\vec{\alpha}$. A reduction is, roughly, obtained by keeping all variables in $\vec{\alpha}$ that are not in $\vec{x}$. This is defined as follows:

Definition 5
Let $\langle N, T, V, P, S \rangle$ be a LCFRS, $\vec{\alpha} \in [(T \cup V)^*]^i$ and $\vec{x} \in V^j$ for some $i, j \in \mathbb{N}$.

Let $w = \vec{\alpha}_1 \$ \dots \$ \vec{\alpha}_i$ be the string obtained form concatenating the components of $\vec{\alpha}$, separated by a new symbol $\$ \notin (V \cup T)$.

Let w' be the image of w under a homomorphism h defined as follows: $h(a) = \$$ for all $a \in T$, $h(X) = \$$ for all $X \in \{\vec{x}_1, \dots \vec{x}_j\}$ and $h(y) = y$ in all other cases.

Let $y_1, \dots y_m \in V^+$ such that $w' \in \$^* y_1 \$^+ y_2 \$^+ \dots \$^+ y_m \$^*$. Then the vector $\langle y_1, \dots y_m \rangle$ is the *reduction* of $\vec{\alpha}$ by $\vec{x}$.

For instance, $\langle aX_1, X_2, bX_3 \rangle$ reduced with $\langle X_2 \rangle$ yields $\langle X_1, X_3 \rangle$ and $\langle aX_1X_2bX_3 \rangle$ reduced with $\langle X_2 \rangle$ yields $\langle X_1, X_3 \rangle$ as well.

The binarization algorithm is given in Fig. 5. As already mentioned, it proceeds like the one for CFGs in the sense that for right-hand sides longer than 2, we introduce a new non-terminal that covers the right-hand side without the first element. Fig. 6 shows an example. In this example, there is only one rule with a right-hand side longer than 2. In a first step, we introduce the new non-terminals and rules that binarize the right-hand side. This leads to the set R. In a second step, before adding the rules from R to the

```
for all rules $r = A(\vec{\alpha}) \rightarrow A_0(\vec{\alpha_0}) \dots A_m(\vec{\alpha_m})$ in $P$ with $m > 1$ do
  remove $r$ from $P$
  $R := \emptyset$
  pick new non-terminals $C_1, \dots, C_{m-1}$
  add the rule $A(\vec{\alpha}) \rightarrow A_0(\vec{\alpha_0})C_1(\vec{\gamma_1})$ to $R$ where $\vec{\gamma_1}$ is obtained by reducing $\vec{\alpha}$ with $\vec{\alpha_0}$
  for all $i$, $1 \leq i \leq m-2$ do
    add the rule $C_i(\vec{\gamma_i}) \rightarrow A_i(\vec{\alpha_i})C_{i+1}(\vec{\gamma_{i+1}})$ to $R$ where $\vec{\gamma_{i+1}}$ is obtained by reducing $\vec{\gamma_i}$ with $\vec{\alpha_i}$
  end for
  add the rule $C_{m-1}(\vec{\gamma_{m-2}}) \rightarrow A_{m-1}(\vec{\alpha_{m-1}})A_m(\vec{\alpha_m})$ to $R$
  for every rule $r' \in R$ do
    replace right-hand side arguments of length $> 1$ with new variables (in both sides) and add the result to $P$
  end for
end for
```

Figure 5
Algorithm for binarizing an LCFRS

grammar, whenever a right-hand side argument contains several variables, these are collapsed into a single new variable.

Original LCFRS:
$S(XYZUVW) \rightarrow A(X,U)B(Y,V)C(Z,W)$
$A(aX,aY) \rightarrow A(X,Y)$ $A(a,a) \rightarrow \varepsilon$
$B(bX,bY) \rightarrow B(X,Y)$ $B(b,b) \rightarrow \varepsilon$
$C(cX,cY) \rightarrow C(X,Y)$ $C(c,c) \rightarrow \varepsilon$

Rule with right-hand side of length > 2: $S(XYZUVW) \rightarrow A(X,U)B(Y,V)C(Z,W)$
For this rule, we obtain
$R = \{S(XYZUVW) \rightarrow A(X,U)C_1(YZ,VW), C_1(YZ,VW) \rightarrow B(Y,V)C(Z,W)\}$

Equivalent binarized LCFRS:
$S(XPUQ) \rightarrow A(X,U)C_1(P,Q)$ $C_1(YZ,VW) \rightarrow B(Y,V)C(Z,W)$
$A(aX,aY) \rightarrow A(X,Y)$ $A(a,a) \rightarrow \varepsilon$
$B(bX,bY) \rightarrow B(X,Y)$ $B(b,b) \rightarrow \varepsilon$
$C(cX,cY) \rightarrow C(X,Y)$ $C(c,c) \rightarrow \varepsilon$

Figure 6
Sample binarization of an LCFRS

The equivalence of the original LCFRS and the binarized grammar is rather straightforward. Note however that the fan-out of the LCFRS can increase.

The binarization depicted in Fig. 5 is deterministic in the sense that for every rule that needs to be binarized, we choose unique new non-terminals. Later, in section 5.3.1, we will introduce additional factorization into the grammar rules that reduces the set of new non-terminals.

3.2 The Parser

As already mentioned, we can assume without loss of generality that our grammars are ε-free and monotone (the grammars that we extract from treebanks all have these properties) and that they contain only binary and unary rules. Furthermore, we assume POS tagging to be done before parsing. POS tags are non-terminals of fan-out 1. Finally, according to our grammar extraction algorithm (see 5.1), a separation between two components always means that there is actually a non-empty gap in between them. Consequently, two different components in a right-hand side can never be adjacent in the same component of the left-hand side. The rules are then either of the form $A(a) \to \varepsilon$ with A a POS tag and $a \in T$ or of the form $A(\vec{x}) \to B(\vec{x})$ or $A(\vec{\alpha}) \to B(\vec{x})C(\vec{y})$ where $\vec{\alpha} \in (V^+)^{dim(A)}$, $\vec{x} \in V^{dim(B)}$, $\vec{y} \in V^{dim(C)}$, i.e., only the rules for POS tags contain terminals in their left-hand sides.

During parsing we have to link the terminals and variables in our LCFRS rules to portions of the input string. These can be characterized by their start and end positions. Therefore we need the notions of ranges, range vectors and rule instantiations. A range is a pair of indices that characterizes the span of a component within the input and a range vector characterizes a tuple in the yield of a non-terminal. A rule instantiation specifies the computation of an element from the lefthand side yield from elements of in the yields of the right-hand side non-terminals based on the corresponding range vectors.

Definition 6 (Range)
Let $w \in T^*$ be a word with $w = w_1 \dots w_n$ where $w_i \in T$ for $1 \leq i \leq n$.

1. $Pos(w) := \{0, \dots, n\}$.
2. We call a pair $\langle l, r \rangle \in Pos(w) \times Pos(w)$ with $l \leq r$ a *range* in w. Its *yield* $\langle l, r \rangle(w)$ is the substring $w_{l+1} \dots w_r$.
3. For two ranges $\rho_1 = \langle l_1, r_1 \rangle, \rho_2 = \langle l_2, r_2 \rangle$, if $r_1 = l_2$, then the concatenation of ρ_1 and ρ_2 is $\rho_1 \cdot \rho_2 = \langle l_1, r_2 \rangle$; otherwise $\rho_1 \cdot \rho_2$ is undefined.
4. A $\vec{\rho} \in (Pos(w) \times Pos(w))^k$ is a k-dimensional *range vector* for w iff $\vec{\rho} = \langle \langle l_1, r_1 \rangle, \dots, \langle l_k, r_k \rangle \rangle$ where $\langle l_i, r_i \rangle$ is a range in w for $1 \leq i \leq k$.

We now define instantiations of rules with respect to a given input string. This definition follows the definition of *clause instantiations* from Boullier (2000). An instantiated rule is a rule in which variables are consistently replaced by ranges. Since we need this definition only for parsing our specific grammars, we restrict ourselves to ε-free rules containing only variables.

Definition 7 (Rule instantiation)
Let $G = (N, T, V, P, S)$ be an ε-free monotone LCFRS. For a given rule $r = A(\vec{\alpha}) \to A_1(\vec{x_1}) \cdots A_m(\vec{x_m}) \in P$ $(0 < m)$ that does not contain any terminals,

1. an *instantiation* with respect to a string $w = t_1 \dots t_n$ consists of a function $f : V \to \{\langle i, j \rangle \mid 1 \leq i \leq j \leq |w|\}$ such that for all x, y adjacent in one of the elements of $\vec{\alpha}$, $f(x) \cdot f(y)$ must be defined; we then define $f(xy) = f(x) \cdot f(y)$,
2. if f is an instantiation of r, then $A(f(\vec{\alpha})) \to A_1(f(\vec{x_1})) \cdots A_m(f(\vec{x_m}))$ is an *instantiated rule* where $f(\langle x_1, \dots, x_k \rangle) = \langle f(x_1), \dots, f(x_k) \rangle$.

We use a probabilistic version of the CYK parser from Seki et al. (1991). The algorithm is formulated using the framework of parsing as deduction (Pereira and Warren 1983; Shieber, Schabes, and Pereira 1995; Sikkel 1997), resp. weighted deductive parsing (Nederhof 2003). In this framework, a set of weighted items representing partial parsing

Scan: $\dfrac{}{0 : [A, \langle\langle i, i+1 \rangle\rangle]}$ A POS tag of w_{i+1}

Unary: $\dfrac{in : [B, \vec{\rho}]}{in + log(p) : [A, \vec{\rho}]}$ $p : A(\vec{\alpha}) \rightarrow B(\vec{\alpha}) \in P$

Binary: $\dfrac{in_B : [B, \vec{\rho_B}], in_C : [C, \vec{\rho_C}]}{in_B + in_C + log(p) : [A, \vec{\rho_A}]}$ $p : A(\vec{\rho_A}) \rightarrow B(\vec{\rho_B})C(\vec{\rho_C})$ is an instantiated rule

Goal: $[S, \langle\langle 0, n \rangle\rangle]$

Figure 7
Weighted CYK deduction system

results is characterized via a set of deduction rules, and certain items (the goal items) represent successful parses.

During parsing, we have to match components in the rules we use with portions of the input string. For a given input w, our items have the form $[A, \vec{\rho}]$ where $A \in N$ and $\vec{\rho}$ is a range vector that characterizes the span of A. Each item has a weight that encodes its probability; we use the absolute value of $log(p)$ where p is the probability.

The first rule (**scan**) tells us that the POS tags that we receive as inputs are given. Consequently, they are axioms and their probability is 1 and their weight therefore 0. The second rule, **unary**, is applied whenever we have found the right-hand side of an instantiation of a unary rule. In our grammar, terminals only occur in rules with POS tags and the grammar is ordered and ε-free. Therefore, the components of the yield of the right-hand side non-terminal and of the left-hand side terminals are the same. The rule **binary** applies an instantiated rule of rank 2. If we already have the two elements of the right-hand side, we can infer the left-hand side element. In both cases, **unary** and **binary**, the probability p of the new rule is multiplied with the probabilities of the antecedent items (which amounts to summing up the antecedent weights and $log(p)$).

Our parser performs a weighted deductive parsing (Nederhof 2003), based on the deduction system from Fig. 7. We use a chart $\mathcal{C}$ and an agenda $\mathcal{A}$, both initially empty, and we proceed as in Fig. 8. Since for all our deduction rules, the weight functions f that compute the weight of a consequent item from the weights of the antecedent items are monotone nonincreasing in each variable, the algorithm from Fig. 8 will always find the best parse without the need of an exhaustive parsing. All new items that we deduce involve at least one of the agenda items as an antecedent item. Therefore, whenever an item is the best in the agenda, we can be sure that we will never find an item with a better (i.e., higher) weight. Consequently, we can safely store this item in the chart and, if it is a goal item, we have found the best parse.

As an example, consider the development of the agenda and the chart in Fig. 10 when parsing aa with the PLCFRS from Fig. 3, transformed into a PLCFRS with preterminals and binarization, i.e., with a POS tag T_a and a new binarization non-terminal B'. The new PLCFRS is given in Fig. 9.

In this example, we find a first analysis for the input (a goal item) when combining an A with span $\langle\langle 0, 2 \rangle\rangle$ into an S. However, this S has a rather low probability and is therefore not on top of the agenda. Later, when finding the better analysis, the weight of the S item in the agenda is updated and then the goal item is the top agenda item and therefore parsing has been successful.

Note that, so far, we have only presented the recognizer. In order to extend it to a parser, we do the following: Whenever we generate a new item, we store it not only with its weight but also with backpointers to its antecedent items. Furthermore, whenever

```
add SCAN results to $\mathcal{A}$
while $\mathcal{A} \neq \emptyset$
  remove best item $x : I$ from $\mathcal{A}$
  add $x : I$ to $\mathcal{C}$
  if $I$ goal item
  then stop and output true
  else
    for all $y : I'$ deduced from $x : I$ and items in $\mathcal{C}$:
      if there is no $z$ with $z : I' \in \mathcal{C} \cup \mathcal{A}$
      then add $y : I'$ to $\mathcal{A}$
      else if $z : I' \in \mathcal{A}$ for some $z$
        then update weight of $I'$ in $\mathcal{A}$ to $max(y, z)$
```

Figure 8
Weighted deductive parsing

PLCFRS with non-terminals $\{S, A, B, T_a\}$, terminals $\{a\}$ and start symbol S:
$0.2 : S(X) \to A(X)$ $0.8 : S(XY) \to B(X, Y)$
$0.7 : A(XY) \to T_a(X)A(Y)$ $0.3 : A(X) \to T_a(X)$
$0.8 : B(ZX, Y) \to T_a(Z)B'(X, Y)$ $1 : B'(X, UY) \to B(X, Y)T_a(U)$
$0.2 : B(X, Y) \to T_a(X)T_a(Y)$ $1 : T_a(a) \to \varepsilon$

Figure 9
Sample binarized PLCFRS (with preterminal T_a)

chart	agenda
	$0 : [T_a, \langle 0, 1\rangle], 0 : [T_a, \langle 1, 2\rangle]$
$0 : [T_a, \langle 0, 1\rangle]$	$0 : [T_a, \langle 1, 2\rangle], 0.5 : [A, \langle 0, 1\rangle]$
$0 : [T_a, \langle 0, 1\rangle], 0 : [T_a, \langle 1, 2\rangle]$	$0.5 : [A, \langle 0, 1\rangle], 0.5 : [A, \langle 1, 2\rangle],$ $0.7 : [B, \langle 0, 1\rangle, \langle 1, 2\rangle]$
$0 : [T_a, \langle 0, 1\rangle], 0 : [T_a, \langle 1, 2\rangle],$ $0.5 : [A, \langle 0, 1\rangle]$	$0.5 : [A, \langle 1, 2\rangle], 0.7 : [B, \langle 0, 1\rangle, \langle 1, 2\rangle],$ $1.2 : [S, \langle 0, 1\rangle]$
$0 : [T_a, \langle 0, 1\rangle], 0 : [T_a, \langle 1, 2\rangle],$ $0.5 : [A, \langle 0, 1\rangle], 0.5 : [A, \langle 1, 2\rangle]$	$0.65 : [A, \langle 0, 2\rangle], 0.7 : [B, \langle 0, 1\rangle, \langle 1, 2\rangle],$ $1.2 : [S, \langle 0, 1\rangle], 1.2 : [S, \langle 1, 2\rangle]$
$0 : [T_a, \langle 0, 1\rangle], 0 : [T_a, \langle 1, 2\rangle],$ $0.5 : [A, \langle 0, 1\rangle], 0.5 : [A, \langle 1, 2\rangle],$ $0.65 : [A, \langle 0, 2\rangle]$	$0.7 : [B, \langle 0, 1\rangle, \langle 1, 2\rangle], 1.2 : [S, \langle 0, 1\rangle],$ $1.2 : [S, \langle 1, 2\rangle], 1.35 : [S, \langle 0, 2\rangle]$
$0 : [T_a, \langle 0, 1\rangle], 0 : [T_a, \langle 1, 2\rangle],$ $0.5 : [A, \langle 0, 1\rangle], 0.5 : [A, \langle 1, 2\rangle],$ $0.65 : [A, \langle 0, 2\rangle], 0.7 : [B, \langle 0, 1\rangle, \langle 1, 2\rangle]$	$0.8 : [S, \langle 0, 2\rangle], 1.2 : [S, \langle 0, 1\rangle],$ $1.2 : [S, \langle 1, 2\rangle]$

Figure 10
Parsing of aa with the grammar from Fig. 3

we update the weight of an item in the agenda, we also update the backpointers. In order to read off the best parse tree, we have to start from the goal item and follow the backpointers.

4. Outside Estimates

So far, the weights we use give us only the inside probability of an item. In order to speed up parsing, we add the estimate of the costs for completing the item into a goal item to its weight. I.e., we add an estimate of the outside probabilities of the items (that is, the absolute value of the logarithm of the estimate) to their weights in the agenda.

All our outside estimates are *admissible* (Klein and Manning 2003a) which means that they never underestimate the actual outside probability of an item. In other words, POS tags: $\dfrac{}{0 : [A, \langle 1 \rangle]}$ A a POS tag Unary: $\dfrac{in : [B, \vec{l}]}{in + log(p) : [A, \vec{l}]}$ $p : A(\vec{\alpha}) \rightarrow B(\vec{\alpha}) \in P$

Binary: $\dfrac{in_B : [B, \vec{l}_B], in_C : [C, \vec{l}_C]}{in_B + in_C + log(p) : [A, \vec{l}_A]}$

where $p : A(\vec{\alpha_A}) \rightarrow B(\vec{\alpha_B})C(\vec{\alpha_C}) \in P$ and the following holds: we define $\mathcal{B}(i)$ as $\{1 \leq j \leq dim(B) \,|\, \vec{\alpha_B}(j)$ occurs in $\vec{\alpha_A}(i)\}$ and $\mathcal{C}(i)$ as $\{1 \leq j \leq dim(C) \,|\, \vec{\alpha_C}(j)$ occurs in $\vec{\alpha_A}(i)\}$. Then for all i, $1 \leq i \leq dim(A)$: $\vec{l}_A(i) = \Sigma_{j \in \mathcal{B}(i)} \vec{l}_B(j) + \Sigma_{j \in \mathcal{C}(i)} \vec{l}_C(j)$.

Figure 11
Inside estimate

they are too optimistic about the costs of completing the item into an S item spanning the entire input.

For the full SX estimate described in section 4.1 and the SX estimate with span and sentence length in section 4.4, the monotonicity is guaranteed and we can do true A* parsing as described by Klein and Manning. The other estimates are not monotonic. This means that it can happen that we deduce an item I_2 from an item I_1 where the weight of I_2 is greater than the weight of I_1. The parser can therefore end up in a local maximum that is not the global maximum we are searching for. In other words, those estimates are only *figures of merit* (FOM).

All outside estimates are computed offline for a certain maximal sentence length len_{max}.

4.1 Full SX estimate

The full SX estimate, for a given sentence length n, is supposed to give the maximal probability of completing a category X with a span ρ into an S with span $\langle\langle 0, n \rangle\rangle$.

For its computation, we need an estimate of the inside probability of a category C with a span ρ, regardless of the actual terminals in our input. This inside estimate is computed as shown in Fig. 11. Here, we do not need to consider the number of terminals outside the span of C (to the left or right or in the gaps), they are not relevant for the inside probability. Therefore the items have the form $[A, \langle l_1, \ldots, l_{dim(A)} \rangle]$, where A is a non-terminal and l_i gives the length of its ith component. It holds that $\Sigma_{1 \leq i \leq dim(A)} l_i \leq len_{max} - dim(A) + 1$ because our grammar extraction algorithm ensures that the different components in the yield of a non-terminal are never adjacent. There is always at least one terminal in between two different components that does not belong to the yield of the non-terminal.

The first rule in Fig. 11 tells us that POS tags always have a single component of length 1, therefore this case has probability 1 (weight 0). The rules **unary** and **binary** are roughly like the ones in the CYK parser, except that they combine items with length information. The rule **unary** for instance tells us that if the log of the probability of building $[B, \vec{l}]$ is greater or equal to in and if there is a rule that allows to deduce an A item from $[B, \vec{l}]$ with probability p, then the log of the probability of $[A, \vec{l}]$ is greater or equal to $in + log(p)$. For each item, we record its maximal weight, i.e., its maximal probability. The rule **binary** is slightly more complicated since we have to compute the length vector of the left-hand side of the rule from the right-hand side length vectors.

A straight-forward extension of the CFG algorithm from Klein and Manning (2003a) for computing the SX estimate is given in Fig. 12. Here, the items have the form

$$\text{Axiom}: \frac{}{0 : [S, \langle 0, \mathit{len}, 0\rangle]} \; 1 \leq \mathit{len} \leq \mathit{len}_{max} \quad \text{Unary:} \frac{w : [A, \vec{l}]}{w + \mathit{log}(p) : [B, \vec{l}]} \; p : A(\vec{\alpha}) \rightarrow B(\vec{\alpha}) \in P$$

$$\text{Binary-right:} \frac{w : [X, \vec{l}_X]}{w + \mathit{in}(A, \vec{l'}_A) + \mathit{log}(p) : [B, \vec{l}_B]}$$

$$\text{Binary-left:} \frac{w : [X, \vec{l}_X]}{w + \mathit{in}(B, \vec{l'}_B) + \mathit{log}(p) : [A, \vec{l}_A]}$$

where, for both binary rules, there is an instantiated rule $p : X(\vec{\rho}) \rightarrow A(\vec{\rho_A})B(\vec{\rho_B})$ such that $\vec{l}_X = l_{out}(\rho), \vec{l}_A = l_{out}(\rho_A), \vec{l'}_A = l_{in}(\rho_A), \vec{l}_B = l_{out}(\rho_B), \vec{l}_B = l_{in}(\rho_B)$.

Figure 12
Full SX estimate top-down

$[A, \vec{l}]$ where the vector $\vec{l}$ tells us about the lengths of the string to the left of the first component, the first component, the string in between the first and second component and so on.

The algorithm proceeds top-down. The outside estimate of completing an S with component length len and no terminals to the left or to the right of the S component (item $[S, \langle 0, \mathit{len}, 0\rangle]$) is 0. If we expand with a unary rule (**unary**), then the outside estimate of the right-hand side item is greater or equal to the outside estimate of the left-hand side item plus the log of the probability of the rule. In the case of binary rules, we have to further add the inside estimate of the other daughter. For this, we need a different length vector (without the lengths of the parts in between the components). Therefore, for a given range vector $\rho = \langle\langle l_1, r_1\rangle, \ldots, \langle l_k, r_k\rangle\rangle$ and a sentence length n, we distinguish between the *inside length vector* $l_{in}(\rho) = \langle r_1 - l_1, \ldots, r_k - l_k\rangle$ and the *outside length vector* $l_{out}(\rho) = \langle l_1, r_1 - l_1, l_2 - r_1, \ldots, l_k - r_{k-1}, r_k - l_k, n - r_k\rangle$.

This algorithm has two major problems: Since it proceeds top-down, in the *Binary* rules, we must compute all splits of the antecedent X span into the spans of A and B which is very expensive. Furthermore, for a category A with a certain number of terminals in the components and the gaps, we compute the lower part of the outside estimate several times, namely for every combination of number of terminals to the left and to the right (first and last element in the outside length vector). In order to avoid these problems, we now abstract away from the lengths of the part to the left and the right, modifying our items such as to allow a bottom-up strategy.

The idea is to compute the weights of items representing the derivations from a certain lower C up to some A (C is a kind of "gap" in the yield of A) while summing up the inside costs of off-spine nodes and the log of the probabilities of the corresponding rules. We use items $[A, C, \rho_A, \rho_C, \mathit{shift}]$ where $A, C \in N$ and ρ_A, ρ_C are range vectors, both with a first component starting at position 0. The integer $\mathit{shift} \leq \mathit{len}_{max}$ tells us how many positions to the right the C span is shifted, compared to the starting position of the A. ρ_A and ρ_C represent the spans of C and A while disregarding the number of terminals to the left the right. I.e., only the lengths of the components and of the gaps are encoded. This means in particular that the length n of the sentence does not play a role here. The right boundary of the last range in the vectors is limited to len_{max}. For any $i, 0 \leq i \leq \mathit{len}_{max}$, and any range vector ρ, we define $\mathit{shift}(\rho, i)$ as the range vector one obtains from adding i to all range boundaries in ρ and $\mathit{shift}(\rho, -i)$ as the range vector one obtains from substracting i from all boundaries in ρ.

$$\text{POS tags: } \frac{}{0 : [C, C, \langle 0,1\rangle, \langle 0,1\rangle, 0]} \quad C \text{ a POS tag}$$

$$\text{Unary: } \frac{0 : [B, B, \rho_B, \rho_B, 0]}{log(p) : [A, B, \rho_B, \rho_B, 0]} \quad p : A(\vec{\alpha}) \rightarrow B(\vec{\alpha}) \in P$$

$$\text{Binary-right: } \frac{0 : [A, A, \rho_A, \rho_A, 0], 0 : [B, B, \rho_B, \rho_B, 0]}{in(A, l_{in}(\rho_A)) + log(p) : [X, B, \rho_X, \rho_B, i]}$$

$$\text{Binary-left: } \frac{0 : [A, A, \rho_A, \rho_A, 0], 0 : [B, B, \rho_B, \rho_B, 0]}{in(B, l_{in}(\rho_B)) + log(p) : [X, A, \rho_X, \rho_A, i]}$$

where i is such that for $shift(\rho_B, i) = \rho'_B$ $p : X(\rho_X) \rightarrow A(\rho_A)B(\rho'_B)$ is an instantiated rule.

$$\text{Starting sub-trees with larger gaps: } \frac{w : [B, C, \rho_B, \rho_C, i]}{0 : [B, B, \rho_B, \rho_B, 0]}$$

$$\text{Transitive closure of sub-tree combination: } \frac{w_1 : [A, B, \rho_A, \rho_B, i], w_2 : [B, C, \rho_B, \rho_C, j]}{w_1 + w_2 : [A, C, \rho_A, \rho_C, i + j]}$$

Figure 13
Full SX estimate bottom-up

The weight of $[A, C, \rho_A, \rho_C, i]$ estimates the log of the probability of completing a C tree with yield ρ_C into an A tree with yield ρ_A such that, if the span of A starts at position j, the span of C starts at position $i + j$. Fig. 13 gives the computation. The value of $in(A, \vec{l})$ is the inside estimate of $[A, \vec{l}]$.

The SX-estimate for some predicate C with span ρ where i is the left boundary of the first component of ρ and with sentence length n is then given by the maximal weight of $[S, C, \langle 0, n\rangle, shift(-i, \rho), i]$.

4.2 SX with Left, Gaps, Right, Length

A problem of the previous estimate is that with a large number of non-terminals, the computation of the estimate requires too much space. Our experiments have shown that for treebank parsing where we have, after binarization and markovization, appr. 12,000 non-terminals, its computation is not feasible. We therefore turn to simpler estimates with only a single non-terminal per item. We now estimate the outside probability of a non-terminal A with a span of a length $length$ (the sum of the lengths of all the components of the span), with $left$ terminals to the left of the first component, $right$ terminals to the right of the last component and $gaps$ terminals in between the components of the A span, i.e., filling the gaps. Our items have the form $[X, len, left, right, gaps]$ with $X \in N$, $len + left + right + gaps \leq len_{max}$, $len \geq dim(X)$, $gaps \geq dim(X) - 1$.

Let us assume that, in the rule $X(\vec{\alpha}) \rightarrow A(\vec{\alpha_A})B(\vec{\alpha_B})$, when looking at the vector $\vec{\alpha}$, we have $left_A$ variables for A-components preceding the first variable of a B component, $right_A$ variables for A-components following the last variable of a B component and $right_B$ variables for B-components following the last variable of a A component. (In our grammars, the first left-hand side argument always starts with the first variable from A.) Furthermore, we set $gaps_A = dim(A) - left_A - right_A$ and $gaps_B = dim(B) - right_B$.

Fig. 14 gives the computation of the estimate. It proceeds top-down, as the computation of the full SX estimate in Fig. 12, except that now the items are simpler. The following side conditions must hold: For *Binary-right* to apply, the following con-

$$\text{Axiom : } \frac{}{0 : [S, len, 0, 0, 0]} \quad 1 \le len \le len_{max}$$

$$\text{Unary: } \frac{w : [X, len, l, r, g]}{w + log(p) : [A, len, l, r, g]} \quad p : X(\vec{\alpha}) \to A(\vec{\alpha}) \in P$$

$$\text{Binary-right: } \frac{w : [X, len, l, r, g]}{w + in(A, len - len_B) + log(p) : [B, len_B, l_B, r_B, g_B]}$$

$$\text{Binary-left: } \frac{w : [X, len, l, r, g]}{w + in(B, len - len_A) + log(p) : [A, len_A, l_A, r_A, g_A]}$$

where, for both binary rules, $p : X(\vec{\alpha}) \to A(\vec{\alpha_A})B(\vec{\alpha_B}) \in P$.

Figure 14
SX with length, left, right, gaps

$$\text{POS tags: } \frac{}{0 : [A, 1]} \quad A \text{ a POS tag} \qquad \text{Unary: } \frac{in : [B, l]}{in + log(p) : [A, l]} \quad p : A(\vec{\alpha}) \to B(\vec{\alpha}) \in P$$

$$\text{Binary: } \frac{in_B : [B, l_B], in_C : [C, l_C]}{in_B + in_C + log(p) : [A, l_B + l_C]}$$

where either $p : A(\vec{\alpha_A}) \to B(\vec{\alpha_B})C(\vec{\alpha_C}) \in P$ or $p : A(\vec{\alpha_A}) \to C(\vec{\alpha_C})B(\vec{\alpha_B}) \in P$.

Figure 15
Inside estimate with total span length

straints must be satisfied: a) $len + l + r + g = len_B + l_B + r_B + g_B$, b) $l_B \ge l + left_A$, c) if $right_A > 0$, then $r_B \ge r + right_A$, else ($right_A = 0$), $r_B = r$, d) $g_B \ge gaps_A$. Similarly, for *Binary-left* to apply, the following constraints must be satisfied: a) $len + l + r + g = len_A + l_A + r_A + g_A$, b) $l_A = l$, c) if $right_B > 0$, then $r_A \ge r + right_B$, else ($right_B = 0$), $r_A = r$ d) $g_A \ge gaps_B$.

The value $in(X, l)$ for a non-terminal X and a length l, $0 \le l \le len_{max}$ is an estimate of the probability of an X category with a span of length l. Its computation is specified in Fig. 15.

The SX-estimate for a sentence length n and for some predicate C with a range characterized by $\vec{\rho} = \langle\langle l_1, r_1\rangle, \dots, \langle l_{dim(C)}, r_{dim(C)}\rangle\rangle$ where $len = \Sigma_{i=1}^{dim(C)}(r_i - l_i)$ and $r = n - r_{dim(C)}$ is then given by the maximal weight of the item $[C, len, l_1, r, n - len - l_1 - r]$.

4.3 SX with LR, Gaps, Length

In order to further decrease the space complexity of the computation of the outside estimate, we can simplify the previous estimate by subsuming the two lengths *left* and *right* in a single length *lr*. I.e., the items now have the form $[X, len, lr, gaps]$ with $X \in N$, $len + lr + gaps \le len_{max}$, $len \ge dim(X)$, $gaps \ge dim(X) - 1$.

The computation is given in Fig. 16. Again, we define $left_A$, $gaps_A$, $right_A$ and $gaps_B$, $right_B$ for a rule $X(\vec{\alpha}) \to A(\vec{\alpha_A})B(\vec{\alpha_B})$ as above. The side conditions are as follows: For *Binary-right* to apply, the following constraints must be satisfied: a) $len + lr + g = len_B + lr_B + g_B$, b) $lr < lr_B$, and c) $g_B \ge gaps_A$. For *Binary-left* to apply, the following must hold: a) $len + lr + g = len_A + lr_A + g_A$, b) if $right_B = 0$ then $lr = lr_A$, else $lr < lr_A$ and c) $g_A \ge gaps_B$.

$$\text{Axiom}: \frac{}{0 : [S, len, 0, 0]} \quad 1 \leq len \leq len_{max}$$

$$\text{Unary:} \frac{w : [X, len, lr, g]}{w + log(p) : [A, len, lr, g]} \quad p : X(\vec{\alpha}) \to A(\vec{\alpha}) \in P$$

$$\text{Binary-right:} \frac{w : [X, len, lr, g]}{w + in(A, len - len_B) + log(p) : [B, len_B, lr_B, g_B]} \quad p : X(\vec{\alpha}) \to A(\vec{\alpha_A})B(\vec{\alpha_B}) \in P$$

$$\text{Binary-left:} \frac{w : [X, len, lr, g]}{w + in(B, len - len_A) + log(p) : [A, len_A, lr_A, g_A]} \quad p : X(\vec{\alpha}) \to A(\vec{\alpha_A})B(\vec{\alpha_B}) \in P$$

Figure 16
SX estimate with length, LR, gaps

$$\text{Axiom}: \frac{}{0 : [S, len, len]} \quad 1 \leq len \leq len_{max}$$

$$\text{Unary:} \frac{w : [X, l_X, slen]}{w + log(p) : [A, l_X, slen]} \quad p : X(\vec{\alpha}) \to A(\vec{\alpha}) \in P$$

$$\text{Binary-right:} \frac{w : [X, l_X, slen]}{w + in(A, l_X - l_B) + log(p) : [B, l_B, slen]} \quad p : X(\vec{\alpha}) \to A(\vec{\alpha_A})B(\vec{\alpha_B}) \in P$$

$$\text{Binary-left:} \frac{w : [X, l_X, slen]}{w + in(B, l_X - l_A) + log(p) : [A, l_A, slen]} \quad p : X(\vec{\alpha}) \to A(\vec{\alpha_A})B(\vec{\alpha_B}) \in P$$

Figure 17
SX estimate with span and sentence length

Furthermore, in both *Binary-left* and *Binary-right*, we have limited lr in the consequent item to the lr of the antecedent plus the length of the sister (len_B, resp. len_A). This results in a further reduction of the number items while having only little effect on the parsing results.

The SX-estimate for a sentence length n and for some predicate C with a span $\vec{\rho} = \langle\langle l_1, r_1\rangle, \ldots, \langle l_{dim(C)}, r_{dim(C)}\rangle\rangle$ where $len = \Sigma_{i=1}^{dim(C)}(r_i - l_i)$ and $r = n - r_{dim(C)}$ is then the maximal weight of $[C, len, l_1 + r, n - len - l_1 - r]$.

4.4 SX with span and sentence length

We will now present a further simplification of the last estimate that records only the span length and the length of the entire sentence. The items have then form $[X, len, slen]$ with $X \in N$, $dim(X) \leq len \leq slen$. The computation is given in Fig. 17. This last estimate is actually monotonic and allows for true A* parsing.

The SX-estimate for a sentence length n and for some predicate C with a span $\vec{\rho} = \langle\langle l_1, r_1\rangle, \ldots, \langle l_{dim(C)}, r_{dim(C)}\rangle\rangle$ where $len = \Sigma_{i=1}^{dim(C)}(r_i - l_i)$ is then the maximal weight of $[C, len, n]$.

In order to prove that this estimate allows for a monotonic weighted deductive parsing and therefore guarantees that the best parse will be found, let us have a look at the CYK deduction rules when being augmented with the estimate. Only *Unary* and *Binary* are relevant since *Scan* does not have antecedent items. The two rules are now as follows:

$$\text{Unary: } \frac{in_B + out_B : [B, \vec{\rho}]}{in_B + log(p) + out_A : [A, \vec{\rho}]} \quad p : A(\vec{\alpha}) \to B(\vec{\alpha}) \in P$$

$$\text{Binary: } \frac{in_B + out_B : [B, \vec{\rho_B}], in_C + out_C : [C, \vec{\rho_C}]}{in_B + in_C + log(p) + out_A : [A, \vec{\rho_A}]} \quad \begin{array}{l} p : A(\vec{\rho_A}) \to B(\vec{\rho_B})C(\vec{\rho_C}) \\ \text{is an instantiated rule} \end{array}$$

(Here, out_A, out_B and out_C are the respective outside estimates of $[A, \vec{\rho_A}]$, $[B, \vec{\rho_B}]$ and $[C, \vec{\rho_C}]$.)

We have to show that for every rule, if this rule has an antecedent item with weight w and a consequent item with weight w', then $w \geq w'$.

Let us start with *Unary*. To show: $in_B + out_B \geq in_B + log(p) + out_A$. Because of the *Unary* rule for computing the outside estimate and because of the unary production, we obtain that, given the outside estimate out_A of $[A, \vec{\rho}]$ the outside estimate out_B of the item $[B, \vec{\rho}]$ is at least $out_A + log(p)$, i.e., $out_B \geq log(p) + out_A$. □

Now let us consider the rule *Binary*. We treat only the relation between the weight of the C antecedent item and the consequent. The treatment of the antecedent B is symmetric. To show: $in_C + out_C \geq in_B + in_C + log(p) + out_A$. Assume that l_B is the length of the components of the B item and n is the sentence length. Then, because of the *Binary-right* rule in the computation of the outside estimate and because of our instantiated rule $p : A(\vec{\rho_A}) \to B(\vec{\rho_B})C(\vec{\rho_C})$, we have that the outside estimate out_C of the C-item is at least $out_A + in(B, l_B) + log(p)$. Furthermore, $in(B, l_B) \geq in_B$. Consequently $out_C \geq in_B + log(p) + out_A$. □

4.5 Integration into the Parser

Before parsing, the outside estimates of all items up to a certain maximal sentence length len_{max} are precomputed. Then, when performing the weighted deductive parsing as explained in section 3.2, whenever a new item is stored in the agenda, we add its outside estimate to its weight.

Since the outside estimate is always greater or equal to the actual outside probability, given the input, the weight of an item in the agenda is always greater or equal to the *log* of the actual product of inside and outside probability of the item. In this sense, the outside estimates given above are admissible.

Additionally, as already mentioned, note that the full SX estimate and the SX estimate with span and sentence length are monotonic and allow for A* parsing. The other two estimates, that are not monotonic, act as FOMs in a Best-First parsing context. Consequently, they contribute to speeding up parsing but they decrease the quality of the parsing output. For further evaluation details see section 6.

5. Grammars for Discontinous Constituents

5.1 Grammar Extraction

The algorithm we use for extracting a LCFRS from a constituency treebank with crossing branches has originally been presented in Maier and Søgaard (2008). It interprets the treebank trees as LCFRS derivation trees. Consider for instance the tree in Fig. 18. The S node has two daughters, a VMFIN node and a VP node. This yields a rule S→VP VMFIN. The VP is discontinuous with two components that wrap around the yield of the VMFIN. Consequently, the LCFRS rule is S(XYZ) →VP(X, Z) VMFIN(Y).

The extraction of an LCFRS from treebanks with crossing branches is almost immediate, except for the fan-out of the non-terminal categories: In the treebank, we can have

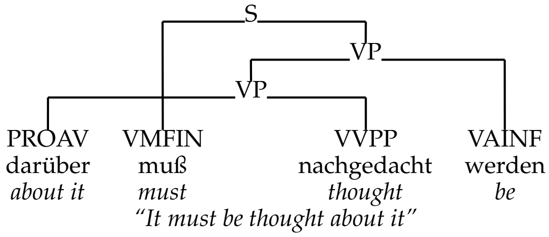


Figure 18
A sample tree from NeGra

the same non-terminal with different arities, for instance a VP without a gap (arity 1), a VP with a single gap (arity 2), and so on. In the corresponding LCFRS, we have to distinguish these different non-terminals. mapping them to different predicates.

The algorithm first creates a so-called lexical clause $P(a) \to \varepsilon$ for each pre-terminal P dominating some terminal a. Then for all other non-terminals A_0 with the children $A_1 \cdots A_m$, a clause $A_0 \to A_1 \cdots A_m$ is created. The numbner of components of the $A_1 \cdots A_m$ is the number of discontinuous parts in their yields. The components of A_0 are concatenations of variables that describe how the discontinuous parts of the yield of A_0 are obtained from the yields of its daughters.

More precisely, the non-terminals in our LCFRS are all A_k where A is a non-terminal label in the treebank and k is a possible fan-out for A. For a given treebank tree $\langle V, E, r, l\rangle$ where V is the set of nodes, $E \subset V \times V$ the set of immediate dominance edges, $r \in V$ the root node and $l : V \to N \cup T$ the labeling function, the algorithm constructs the following rules. Let us assume that $w_1, \dots, w_n$ are the terminal labels of the leaves in $\langle V, E, r\rangle$ with a linear precedence relation $w_i \prec w_j$ for $1 \leq i < j \leq n$. We introduce a variable X_i for every w_i, $1 \leq i \leq n$.

- For every pair of nodes $v_1, v_2 \in V$ with $\langle v_2, v_2\rangle \in E$, $l(v_2) \in T$, we add $l(v_1)(l(v_2)) \to \varepsilon$ to the rules of the grammar. (We omit the fan-out subscript here since pre-terminals are always of fan-out 1.)
- For every node $v \in V$ with $l(v) = A_0 \notin T$ such that there are exactly m nodes $v_1, \dots, v_m \in V$ ($m \geq 1$) with $\langle v, v_i\rangle \in E$ and $l(v_i) = A_i \notin T$ for all $1 \leq i \leq m$, we now create a rule

$$A_0(x_1^{(0)}, \dots, x_{dim(A_0)}^{(0)}) \\ \to A_1(x_1^{(1)}, \dots, x_{dim(A_1)}^{(1)}) \dots A_m(x_1^{(m)}, \dots, x_{dim(A_m)}^{(m)})$$

  where for the predicate A_i, $0 \leq i \leq m$, the following must hold:

  1. The concatenation of all arguments of A_i, $x_1^{(i)} \dots x_{dim(A_i)}^{(i)}$ is the concatenation of all $X \in \{X_i \mid \langle v_i, v_i'\rangle \in E^* \text{ with } l(v_i') = w_i\}$ such that X_i precedes X_j if $i < j$, and
  2. a variable X_j with $1 \leq j < n$ is the right boundary of an argument of A_i if and only if $X_{j+1} \notin \{X_i \mid \langle v_i, v_i'\rangle \in E^* \text{ with } l(v_i') = w_i\}$, i.e., an argument boundary is introduced at each discontinuity.

  As a further step, in this new rule, all right-hand side arguments of length > 1 are replaced in both sides of the rule with a single new variable.

$$\begin{array}{rcl}
\text{PROAV(Darüber)} & \rightarrow & \varepsilon \\
\text{VMFIN(muß)} & \rightarrow & \varepsilon \\
\text{VVPP(nachgedacht)} & \rightarrow & \varepsilon \\
\text{VAINF(werden)} & \rightarrow & \varepsilon \\
\text{S}_1(X_1X_2X_3) & \rightarrow & \text{VP}_2(X_1,X_3)\ \text{VMFIN}(X_2) \\
\text{VP}_2(X_1,X_2X_3) & \rightarrow & \text{VP}_2(X_1,X_2)\ \text{VAINF}(X_3) \\
\text{VP}_2(X_1,X_2) & \rightarrow & \text{PROAV}(X_1)\ \text{VVPP}(X_2)
\end{array}$$

Figure 19
LCFRS rules extracted from the tree in Fig. 18

> Finally, all non-terminals A in the rule are equipped with an additional subscript $dim(A)$ which gives us the final non-terminal in our LCFRS.

For the tree in Fig. 18, the algorithm produces for instance the rules in Fig. 19.

The probabilities are then computed based on the frequencies of rules in the treebank, using a Maximum Likelihood estimator (MLE). Such an estimation has been used before (Levy 2005; Kato, Seki, and Kasami 2006).

5.2 Binarization

In LCFRS, in contrast to CFG, the order of the right-hand side elements of a rule does not matter for the result of a derivation. Therefore, we can reorder the right-hand side of a rule before binarizing it. We employ different re-orderings.

5.2.1 Using Head Rules. The first re-ordering results in a head-outward binarization where the head is the lowest subtree and it is extended by adding first all sisters to its left and then all sisters to its right. Consequently, before binarizing we reorder the right-hand side of the rules extracted from the treebank such that first, all elements to the right of the head are listed in reverse order, then all elements to the left of the head in their original order and then the head itself. We also use a variant of this re-ordering where we add first the sisters to the right and then the ones to the left. This is what Klein and Manning (2003b) do.

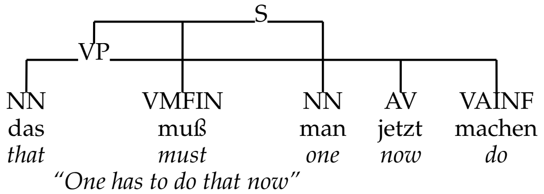
We apply Collins-style head rules, based on the rules the Stanford parser (Klein and Manning 2003c) uses for NeGra, to mark the respective head daughters of all non-terminal nodes.

5.2.2 Minimizing Fan-Out and Arity. We furthermore use another binarization order that yields a minimal arity and a minimal variable number per production and binarization step. Gómez-Rodríguez et al. (2009) have shown how to obtain such an optimal binarization for a given LCFRS.[2] We assume that we are only considering partitions of right-hand sides where one of the sets contains only a single non-terminal.

For a given rule $r = A_0(\vec{x_0}) \rightarrow A_1(\vec{x_1}) \dots A_k(\vec{x_k})$, we define the *characteristic string* $s(c, A_i)$ of the A_i-reduction of c as follows: Concatenate the elements of $\vec{x_0}$, separated with new additional symbols \$ while replacing every component from $\vec{x_i}$ with a \$. We then define the arity of the characteristic string, $dim(s(c, A_i))$, as the number of maximal substrings $x \in V^+$ in $s(A_i)$.

[2] This version of the algorithm has first been published in Kallmeyer (2010).

Tree in NeGra Format:



Rule extracted for the S node: S$(XYZU) \rightarrow$ VP(X, U) VMFIN(Y) NN(Z)
Reordering for head-outward binarization: S$(XYZU) \rightarrow$ NN(Z) VP(X, U) VMFIN(Y)
New rules resulting form binarizing this rule:
S$(X) \rightarrow \rightarrow$ S$_{bin1}(X, Z)$ NN(Y)
S$_{bin2}(X) \rightarrow$ VMFIN(X) S$_{bin1}(XY, Z) \rightarrow$ VP(X, Z) S$_{bin2}(Y)$

Tree after binarization:

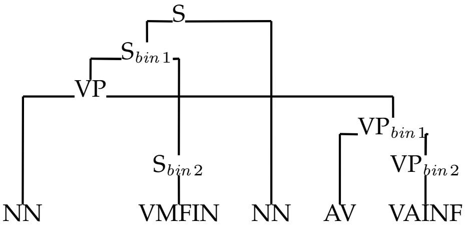


Figure 20
Sample head-outward binarization

Fig. 21 shows how in a first step, for a given rule r with right-hand side length > 2, we determine the optimal candidate for binarization: On all right-hand side predicates B we check for the maximal arity (given by $dim(s(c, B))$) and the number of variables ($dim(s(c, B)) + dim(B)$) we would obtain when binarizing with this predicate. This check provides the optimal candidate. In a second step we then perform the same binarization as before, except that we use the optimal candidate now instead of the first element of the right-hand side.

```
$cand = 0$
$arity$ = number of variables in $r$
$vars$ = number of variables in $r$
**for all** $i = 0$ **to** $m$ **do**
    $cand\text{-}arity = dim(s(r, A_i))$;
    **if** $cand\text{-}arity < arity$ and $dim(A_i) < arity$ **then**
        $arity = max(\{cand\text{-}arity, dim(A_i)\})$;
        $vars = cand\text{-}arity + dim(A_i)$;
        $cand = i$;
    **else if** $cand\text{-}arity \leq arity, dim(A_i) \leq arity$ and $cand\text{-}arity + dim(A_i) < vars$ **then**
        $arity = max(\{cand\text{-}arity, dim(A_i)\})$;
        $vars = cand\text{-}arity + dim(A_i)$;
        $cand = i$
    **end if**
**end for**
```

Figure 21
Optimized version of the binarization algorithm, determining binarization order

5.2.3 Unary rules. In all binarizations, we add additional unary rules when deriving the head. This allows for a further factorization that has proved itself useful in parsing.

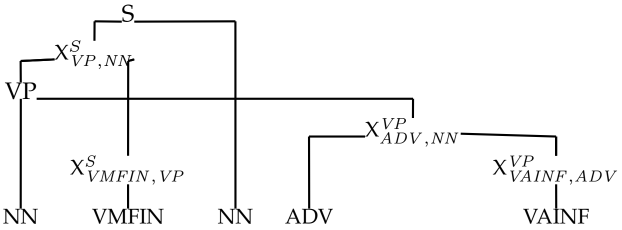


Figure 22
Sample Markovization with $v = 1, h = 2$

Fig. 20 shows a sample binarization of a tree in the NeGra format. We do not insert an additional unary rule at the highest new binarization non-terminal, since this was neither beneficial for parsing times nor for the parsing results.

5.3 Incorporating Additional Context

5.3.1 Markovization. As already mentioned in section 3.1, a binarization that introduces unique new non-terminals for every single rule that needs to be binarized, produces a large amount of non-terminals and fails to capture certain generalizations. For this reason, we introduce *markovization* (Collins 1999).

Markovization is achieved by introducing only a single new non-terminal for the new rules introduced during binarization and adding vertical and horizontal context from the original trees to each occurrence of this new non-terminal. As vertical context, we add the first v labels on the path from the root node of the tree that we want to binarize to the root of the entire treebank tree. The vertical context is collected during grammar extraction and then taken into account during binarization of the rules. As horizontal context, during binarization of a rule $A(\vec{\alpha}) \rightarrow A_0(\vec{\alpha_0}) \dots A_m(\vec{\alpha_m})$, for the new non-terminal that comprises the right-hand side elements $A_i \dots A_m$ (for some $1 \leq i \leq m$), we add the first h elements of $A_i, A_{i-1}, \dots, A_0$.

Figure 22 shows an example of a markovization of the tree from Fig. 20 with $v = 1$ and $h = 2$. Here, the superscript is the vertical context and the subscript the horizontal context of the new non-terminal X. Note that in this example, we have disregarded the fan-out of the context categories. The VP for instance is actually a VP$_2$ since it has fan-out 2. For the context symbols, one can either use the categories from the original treebank (without fan-out) or the ones from the LCFRS rules (with fan-out). We chose the latter approach.

Note finally that the markovization is independent of the binarization strategy.

5.3.2 Further Category Splitting. Grammar annotation (i.e., manual enhancement of annotation information through category splitting) has previously been successfully employed in parsing German (Versley 2005). In order to see if such modifications can have a beneficial effect in PLCFRS parsing as well, we perform different category splits on the (unbinarized) NeGra constituency data.

We split the category S (“sentence”) into SRC (“relative clause”) and S (all other categories S). Relative clauses mostly occur in a very specific context, namely as the right part of an NP or a PP. This splitting should therefore speed up parsing and increase precision. Furthermore, we distinguish NPs by their case. More precisely, to all nodes with categories N, we append the grammatical function label to the category label. We finally experiment with the combination of both splits.

	training	test
number of sentences	16,502	1,833
av. sentence length	14.56	14.62
av. tree height	4.62	4.72
av. children per node	2.96	2.94
sentences without gaps	12,481 (75.63%)	1,361 (74.25%)
sent. with one gap	3,320 (20.12%)	387 (21.11%)
sent. with $\geq$ 2 gaps	701 (4.25%)	85 (4.64%)
max. gap degree	6	5
wellnested sentences	16,145 (97.84%)	1,792 (97.76%)
1-ill-nested sentences	346 (2.10%)	40 (2.24%)
$\geq$ 2-ill-nested sentences	11 (0.07%)	1 (0.05%)

Table 1
NeGra: Properties of the data with crossing branches

6. Experiments

6.1 Data

Our data source is the NeGra treebank (Skut et al. 1997). We create two different data sets for constituency parsing. For the first one, we start out with the unmodified NeGra treebank and remove all sentences with a length of more than 30 words. We preprocess the treebank following common practice (Kübler and Penn 2008), attaching all nodes which are attached to the virtual root node to nodes within the tree such that, ideally, no new crossing edges are created. In a second pass, we attach punctuation which comes in pairs (parentheses, quotation marks) to the same nodes. For the second data set we create a copy of the preprocessed first data set, in which we apply the usual tree transformations for NeGra PCFG parsing, i.e., moving nodes to higher positions until all crossing branches are resolved. The first 90% of both data sets are used as the training set and the remaining 10% as test set.

Tab. 1 lists some properties of the training and test (resp. gold) sets with crossing branches, namely the total number of sentences, the average sentence length, the average tree height, the height of a tree being the length of the longest of all paths from the terminals to the root node, and the average number of children per node (excluding terminals). Furthermore, gap degree and well-nestedness are listed (Maier and Lichte 2009).

Our findings correspond to those of Maier and Lichte. However, we have sentences which are $\geq$ 2-ill-nested, while they only found 1-ill-nested sentences. This is due to punctuation attachment; unlike us, Maier and Lichte removed the punctuation from the trees.

6.2 Parser Implementation

We have implemented the CYK parser described in the previous section in a system called `rparse`. It is available under the GNU General Public License 2.0 at `http://www.sfs.uni-tuebingen.de/emmy/rparse`. The implementation is realized in Java.

6.3 Evaluation

For the evaluation of the constituency parses, we use an EVALB-style metric. For a tree over a string w, a single constituency is represented by a tuple $\langle A, \vec{\rho}\rangle$ with A a node label and $\vec{\rho} \in (Pos(w) \times Pos(w))^{dim(A)}$.[3] We compute precision, recall and F_1 based on these tuples from gold and parsed test data. Despite the shortcomings of such a measure (Rehbein and van Genabith 2007, e.g.), it still allows to some extent a comparison to previous work in PCFG parsing. For a more detailed evaluation of NeGra PLCFRS constituency parsing results, see Maier (2010).

6.4 Markovization and Binarization

We use the markovization settings $v = 1$ and $h = 2$ for all further experiments. The setting which has been reported to yield the best results for PCFG parsing of NeGra, $v = 2$ and $h = 1$ (Rafferty and Manning 2008), required a parsing time which was too high.[4]

Tab. 2 contains the parsing results for the LCFRS data set using five different binarizations: *Headdriven* and *KM* are the two head-outward binarizations which use a head chosen on linguistic grounds (described in 5.2.1), *L-to-R* is another variant in which we always choose the rightmost daughter of a node as its head.[5] *Optimal* re-orders the left-hand side such that the fan-out of the binarized rules is optimized (described in 5.2.2). Finally, we also try a deterministic binarization (*Deterministic*) in which we binarize strictly from left to right, i.e., we do not reorder the right-hand sides of productions, and choose unique binarization labels.

	Headdriven	KM	L-to-R	Optimal	Deterministic
LP	73.90	74.25	74.86	74.66	72.54
LR	74.55	74.75	74.98	75.12	71.93
LF_1	74.22	74.49	74.92	74.89	72.23
UP	77.17	77.68	77.81	77.67	75.99
UR	77.85	78.20	77.93	78.15	75.35
UF_1	77.51	77.94	77.87	77.91	75.67

Table 2
NeGra: LCFRS results for different binarizations

The results of the headdriven binarizations and the optimal binarization lie close together, the results for the deterministic binarization are worse. This indicates that the presence or absence of markovization has more impact on parsing results than the actual binarization order. Furthermore, the non-optimal binarizations did not yield a binarized grammar of a higher arity than the optimal binarization: For all five binarizations, the arity was 7 (caused by a VP interrupted by punctuation).

[3] Note that our metric is equivalent to the corresponding PCFG metric for $dim(A) = 1$.

[4] Older versions of rparse contained a bug which kept the priority queue from being updated correctly, i.e., during an update, the corresponding node in the priority queue was not moved to its top, and therefore the best parse was not guaranteed to be found; however, higher parsing speeds were achieved. The current version of rparse implements the update operation correctly, using a Fibonacci queue to ensure efficiency (Cormen et al. 2003).

[5] The term *head* is not used in its proper linguistic sense here.

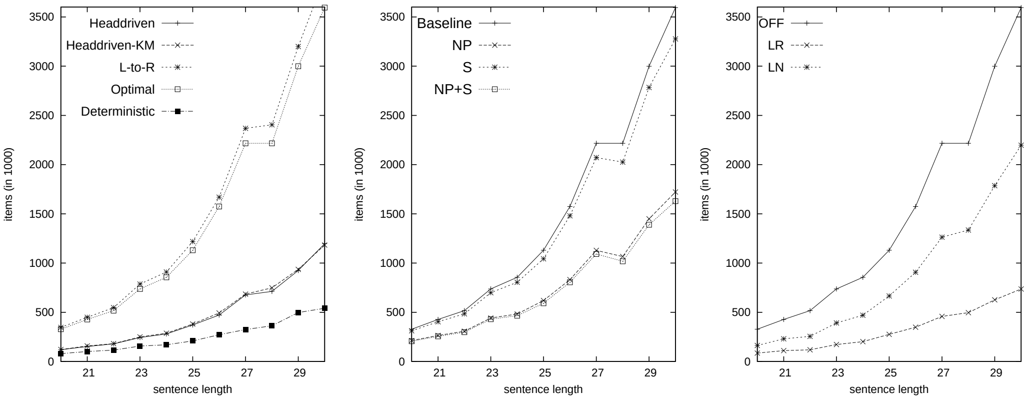


Figure 23
NeGra: Items for LCFRS parsing (left-to-right): binarizations, baseline and category splits, estimates

The different binarizations result in different numbers of items, and therefore allow for different parsing speeds. In Fig. 23 a visual representation of the number of items produced by all binarizations is shown (on the left). Note that when choosing the head with head rules the number of items is almost not affected from the choice of adding first the children to the left of the head and then to the right of the head or vice versa. We will use the optimal binarization in all further experiments.

6.5 Baseline Evaluation and Category Splits

		w/ category splits				w/ category splits		
	LCFRS	*NP*	*S*	*NP* ∘ *S*	PCFG	*NP*	*S*	*NP* ∘ *S*
LP	74.66	74.97	75.48	75.83	76.49	77.04	77.13	77.68
LR	75.12	75.36	75.89	76.16	76.07	76.92	76.79	77.74
LF_1	74.89	75.16	75.68	76.00	76.28	76.98	76.96	77.71
UP	77.67	78.00	78.20	78.65	79.22	79.70	79.81	80.27
UR	78.15	78.41	78.63	78.99	78.79	79.58	79.46	80.33
UF_1	77.91	78.20	78.41	78.82	79.00	79.64	79.64	80.30

Table 3
NeGra: LCFRS and PCFG baseline and category splits

Tab. 3 presents the constituency parsing results for the LCFRS data set (with crossing branches) and the PCFG data set (without crossing branches), both with and without the different category splits. We evaluate the parser output against the unmodified gold data, i.e., before we evaluate the experiments with category splits, we replace all split labels in the parser output with the corresponding original labels.

We see that the quality of the LCFRS parser output (which contains more information than the output of a PCFG parser) does not lag far behind the quality of the PCFG parsing results. With respect to the category splits, the results show furthermore that category splitting is indeed beneficial for the quality of the PLCFRS parser output. Interestingly, the S split is only beneficial for PLCFRS parsing. This is due to the fact that the characteristic context in which relative clauses occur disappears when removing the crossing branches (relative clauses are attached high). The gains in speed are particularly visible for sentences with a length greater than 20 words (cf. the number of produced items in Fig. 23 (middle)).

We take a closer look at the properties of the trees in the parser output for the LCFRS set. 12 sentences had no parse, therefore, the parser output has 1,821 sentences. The average tree height is 4.71, and the average number of children per node (excluding terminals) is 2.92. These values are almost identical to the values for the gold data. As for the gap degree, we get 1,394 sentences with no gaps, 359 with gap degree 1 and 69 with 2 or 3 gaps. Even though the difference is only small, one can see that fewer gaps are preferred. This is not surprising, since constituents with many gaps are rare events and therefore end up with a probability which is too low. 1,803 sentences in the parser output are analyzed as well-nested, and the remaining 18 as 1-ill-nested. The fact that almost the same amount of sentences is well-nested in the gold data and in the parser output is misleading; only 3 of the ill-nested sentences from the gold data are also analyzed as ill-nested by the parser. This also unsurprisingly indicates sparse data problems.

6.6 Evaluating outside estimates

We compare the parser performance without estimates (OFF) with its performance with the estimates described in 4.3 (LR) and 4.4 (LN). Unfortunately, the full estimates are too expensive to compute both in terms of time and space. Unlike the LN estimate, which allows for true A* parsing, the LR estimate lets the quality of the parsing results deteriorate slightly – compared to the baseline, labeled F$_1$ drops from 74.89 to 73.62 and unlabeled F$_1$ drops from 77.91 to 76.89. Fig. 23 (right) shows the average number of items produced by the parser for different sentence lengths. The results indicate that the estimates have the desired effect of preventing unnecessary items from being produced. This is reflected in a significantly lower parsing time.

7. Comparison to other Approaches

Our results are not directly comparable with PCFG parsing results, since LCFRS parsing is a harder task. However, since the EVALB metric coincides for constituents without crossing branches, in order to place our results in the context of previous work on parsing NeGra, we cite some of the results from the literature which were obtained using PCFG parsers[6]: Kübler (2005) (Tab. 1, plain PCFG) obtains 69.4, Dubey and Keller (2003) (Tab. 5, sister-head PCFG model) 71.12, Rafferty and Manning (2008) (Tab. 2, Stanford parser with markovization $v = 2$ and $h = 1$) 77.2, and Petrov and Klein (2007) (Tab. 1, Berkeley parser) 80.1.

As for the works which aim at creating crossing branches, Plaehn (2004) obtains 73.16 Labeled F_1 using Probabilistic Discontinuous Phrase Structure Grammar (DPSG), albeit only on sentences with a length of up to 15 words. On those sentences, we obtain 84.59. Levy (2005) does not provide any evaluation results of his work. Very recently, Evang and Kallmeyer (2011) followed up on our work. They transform the Penn Treebank such that the trace nodes and co-indexation are converted in crossing branches

[6] Note that these results were obtained on sentences with a length of ≤ 40 words and that those parser possibly would deliver better results if tested on our test set.

and parse them with the parser presented in this article, obtaining promising results. van Cranenburgh, Scha, and Sangati (2011) have also followed up on our work. They introduce an integration of our approach with Data-Oriented Parsing (DOP), producing results which slightly exceed ours.

The comparison shows that our system delivers competitive results. Additionally, when comparing this to PCFG parsing results, one has to keep in mind that LCFRS parse trees contain non-context-free information about discontinuities. Therefore, a correct parse with our grammar is actually better than a correct CFG parse, evaluated with respect to a transformation of NeGra into a context-free treebank where precisely this information gets lost.

8. Conclusion

We have presented the first efficient implementation of a weighted deductive CYK parser for Probabilistic Linear Context-Free Rewriting Systems (PLCFRS), showing that LCFRS indeed allows for data-driven parsing while modeling discontinuities in a straightforward way. To speed up parsing, we have introduced different context-summary estimates of parse items, some acting as Figures-Of-Merit, others allowing for A^* parsing. We have implemented the parser and we have evaluated it with grammars extracted from the German NeGra treebank. Our experiments show that data-driven LCFRS parsing is feasible and yields output of competitive quality.

There are two main directions for future work on this subject.

- On the symbolic side, LCFRS seems to offer more power than necessary. By removing symbolic expressivity, a lower parsing complexity can be achieved. One possibility is to refrain from using ill-nestedness. See to this respect the parsing algorithm for well-nested LCFRS of Gómez-Rodríguez, Kuhlmann, and Satta (2010). Nevertheless, ill-nestedness has been shown to be a necessary tool for linguistic modeling (Chen-Main and Joshi 2010), even though only about 3% of the treebank sentences are analyzed as illnested. Our goal for future work is therefore to find a reduced variant of illnestedness with which we get a lower parsing complexity.
- On the side of the probabilistic model, there are certain independence assumptions made in our model that are too strong. The main problem in this respect is that, due to the definition of LCFRS, we have to distinguish between occurrences of the same category with different fan-outs. For instance, VP_1 (no gaps), VP_2 (one gap), etc. are different non-terminals. Consequently, the way they expand are considered independent from each other. This, however, is of course not true. Furthermore, some of these non-terminals are rather rare, we therefore have a sparse data problem here. This leads to the idea to separate between the development of a category (independent from its fan-out) and the fan-out and position of gaps. We plan to integrate this into our probabilistic model in future work.

Acknowledgments
We are particularly grateful to Giorgio Satta for extensive discussions of the details of the probabilistic treebank model presented in this paper. Furthermore, we owe a lot to Kilian Evang who participated in the implementation of the parser.
While writing this paper, we received funding from the German Research Foundation DFG (Deutsche Forschungsgemeinschaft) in the form of an Emmy Noether Grant.

References

Boullier, Pierre. 1998a. 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, University of Pennsylvania, Philadelphia.

Boullier, Pierre. 1998b. A Proposal for a Natural Language Processing Syntactic Backbone. Technical Report 3342, INRIA.

Boullier, Pierre. 2000. Range Concatenation Grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT2000)*, pages 53–64, Trento, Italy, February.

Boyd, Adriane. 2007. Discontinuity revisited: An improved conversion to context-free representations. In *The Linguistic Annotation Workshop at ACL 2007*, Prague, Czech Republic.

Charniak, Eugene and Sharon A. Caraballo. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24.

Chen-Main, Joan and Aravind Joshi. 2010. Unavoidable ill-nestedness in natural language and the adequacy of tree local-MCTAG induced dependency structures. In *Proceedings of the Tenth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+10)*.

Chiang, David. 2003. Statistical parsing with an automatically extracted Tree Adjoining Grammar. In *Data-Oriented Parsing*. CSLI Publications.

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

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2003. *Introduction to Algorithms*. MIT Press, Cambridge, 2nd edition. 4th printing.

Dienes, Péter. 2003. *Statistical Parsing with Non-local Dependencies*. Ph.D. thesis, Saarland University.

Dubey, Amit and Frank Keller. 2003. Probabilistic parsing for German using sisterhead dependencies. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, Sapporo, Japan.

Evang, Kilian and Laura Kallmeyer. 2011. PLCFRS parsing of english discontinuous constituents. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT 2011)*, Dublin, Ireland. To appear.

Gómez-Rodríguez, Carlos, Marco Kuhlmann, and Giorgio 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 Association for Computational Linguistics*, pages 276–284, Los Angeles, CA.

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

Hockenmaier, Julia. 2003. *Data and models for Statistical Parsing with Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh.

Johnson, Mark. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, PA.

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

Kallmeyer, Laura and Wolfgang Maier. 2010. Data-driven parsing with probabilistic Linear Context-Free Rewriting Systems. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, Beijing, China.

Kato, Yuki, Hiroyuki Seki, and Tadao Kasami. 2006. Stochastic multiple context-free grammar for rna pseudoknot modeling. In *Proceedings of The Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+8)*, Sydney, Australia.

Klein, Dan 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*, Edmonton, Canada.

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

Klein, Dan and Christopher D. Manning. 2003c. Fast exact inference with a factored model for natural language parsing. In *In Advances in Neural Information Processing Systems 15 (NIPS)*. MIT Press.

Kracht, Marcus. 2003. *The Mathematics of Language*. Number 63 in Studies in Generative Grammar. Mouton de Gruyter, Berlin.

Kübler, Sandra. 2005. How do treebank annotation schemes influence parsing results? Or how not to compare apples and oranges. In *Recent Advances in Natural Language Processing 2005 (RANLP 2005)*, Borovets, Bulgaria.

Kübler, Sandra and Gerald Penn, editors. 2008. *Proceedings of the Workshop on Parsing German at ACL 2008*. Association for Computational Linguistics, Columbus, Ohio.

Kuhlmann, Marco and Giorgio Satta. 2009. Treebank grammar techniques for non-projective dependency parsing. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, Athens, Greece.

Levy, Roger. 2005. *Probabilistic Models of Word Order and Syntactic Discontinuity*. Ph.D. thesis, Stanford University.

Levy, Roger and Christopher D. Manning. 2004. Deep dependencies from context-free statistical parsers: correcting the surface dependency approximation. In *Proceedings of ACL*, Barcelona, Spain.

Maier, Wolfgang. 2010. Direct parsing of discontinuous constituents in German. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 58–66, Los Angeles, CA, USA, June. Association for Computational Linguistics.

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

Maier, Wolfgang and Timm Lichte. 2009. Characterizing Discontinuity in Constituent Treebanks. In *Proceedings of the 14th Conference on Formal Grammar 2009*, Bordeaux, France.

Maier, Wolfgang and Anders Søgaard. 2008. Treebanks and mild context-sensitivity. In *Proceedings of the 13th Conference on Formal Grammar 2008*, Hamburg, Germany.

Marcus, Mitchell, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: annotating predicate argument structure. In *Proceedings of HLT*.

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

Nederhof, Mark-Jan. 2003. Weighted Deductive Parsing and Knuth's Algorithm. *Computational Linguistics*, 29(1).

Pereira, Fernando C. N. and David Warren. 1983. Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144, Cambridge, MA.

Petrov, Slav and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, Rochester, NY.

Plaehn, Oliver. 2004. Computing the most probable parse for a discontinuous phrase-structure grammar. In *New developments in parsing technology*. Kluwer.

Rafferty, Anna and Christopher D. Manning, 2008. *Parsing Three German Treebanks: Lexicalized and Unlexicalized Baselines*. In Kübler and Penn (2008).

Rehbein, Ines and Josef van Genabith. 2007. Evaluating evaluation measures. In *Proceedings of NODALIDA 2007*, Tartu, Estonia.

Seddah, Djame, Sandra Kübler, and Reut Tsarfaty, editors. 2010. *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*. Association for Computational Linguistics, Los Angeles, CA.

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

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

Sikkel, Klaas. 1997. *Parsing Schemata*. Texts in Theoretical Computer Science. Springer, Berlin, Heidelberg, New York.

Skut, Wojciech, Brigitte Krenn, Thorten Brants, and Hans Uszkoreit. 1997. An Annotation Scheme for Free Word Order Languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP)*, Washington, D.C.

van Cranenburgh, Andreas, 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)*, Dublin, Ireland. To appear.

Versley, Yannick. 2005. Parser evaluation across text types. In *Proceedings of TLT 2005*.

Vijay-Shanker, K., David J. Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford.